d.velop

d.velop process studio: Administrieren

Inhaltsverzeichnis

1. Administrationshandbuch d.velop process studio

1.1. Basisinformationen zur Anwendung

In diesem Kapitel finden Sie Produkthinweise und allgemeine Informationen.

1.1.1. Über d.velop process studio

d.velop process studio bildet den zentralen Einstieg für das Thema Prozessautomatisierung. d.velop process studio besteht aus mehreren Komponenten und Werkzeugen, die Sie dabei unterstützen Ihren Arbeitsalltag zu digitalisieren, automatisieren und dokumentieren.

Mit d.velop process studio sind Sie in der Lage, Geschäftsprozesse zu automatisieren. Sie können Anwender in Form von Aufgaben an Prozessen teilnehmen lassen oder automatisierte Serviceaktivitäten ausführen. d.velop process studio unterstützt den BPMN-Standard. Dadurch haben Sie vielfältige Möglichkeiten bei der Gestaltung von Prozessen.

In der Prozessüberwachung haben Sie die Möglichkeit, den Status aktiver Prozesse zu kontrollieren. Im Falle eines Fehlers haben Sie direkt die Möglichkeit, Korrekturen vorzunehmen. Damit stellen Sie auf einfache Weise den erfolgreichen Betrieb Ihrer Geschäftsprozesse sicher.

Die Prozessadministration unterstützt Sie dabei, Aktionen über mehrere Prozessinstanzen durchzuführen. Sie können beispielsweise mehrere Prozessinstanzen abbrechen, diese bei Prozessänderungen in eine neue Version migrieren, fehlerhafte Prozessinstanzen erneut ausführen oder eine Prozessversion löschen.

Wenn Sie einen eigenen Prozess in der Prozessverwaltung modelliert haben, steht Ihnen dieser innerhalb der d.velop platform zur Verfügung.

1.2. Installieren und Deinstallieren

In diesem Thema finden Sie Informationen zur Installation von d.velop process studio.

1.2.1. Systemvoraussetzungen

Beachten Sie die allgemeinen Systemvoraussetzungen für d.velop-Produkte.

1.2.2. Installieren von d.velop process studio

Sie installieren die Software ausschließlich mit d.velop software manager. Wird eine Anwendung für verschiedene Produkte benötigt, werden die zugehörigen Softwarepakete ebenfalls automatisch installiert.

Weitere Informationen zum Installieren der Software finden Sie im d.velop software manager-Handbuch.

1.2.3. Deinstallieren von d.velop process studio

Sie können die Software, die Sie mit d.velop software manager installiert haben, nur mit d.velop software manager wieder deinstallieren. Falls es bei der zu deinstallierenden Software Abhängigkeiten zu anderen Softwarepaketen gibt, müssen Sie diese Konflikte entsprechend auflösen.

Weitere Informationen zum Deinstallieren finden Sie im d.velop software manager-Handbuch.

1.2.4. Installieren von Updates für d.velop process studio

Sie aktualisieren die Software nur noch mit d.velop software manager.

Weitere Informationen zum Aktualisieren finden Sie im d.velop software manager-Handbuch.

1.2.5. Rollback einer Installation von d.velop process studio

Sie können von der Software, die Sie mit d.velop software manager installiert haben, eine frühere Version wiederherstellen. Bei diesem Prozess wird die Software nur auf eine vorherige Version zurückgesetzt.

Weitere Informationen zur Wiederherstellung einer früheren Version finden Sie im d.velop software manager-Handbuch.

1.2.6. Freigeben der Standardports für d.velop process studio

- Standardmäßig wird der Port für d.velop process dynamisch ermittelt. Sie können jedoch auch einen Port festlegen.

1.3. Konfigurieren von d.velop process studio

In diesem Thema finden Sie Informationen zur Konfiguration und weiteren Einstellungen.

1.3.1. Konfigurieren von d.velop process

In diesem Thema finden Sie Informationen zur Konfiguration von d.velop process.

Konfigurieren der Datenbank

Sie benötigen eine Datenbank für die Anwendung. Es wird vorausgesetzt, dass Sie bereits eine Datenbank angelegt haben, in der die Anwendung Daten speichern soll. Zusätzlich sind zwei Benutzer notwendig, mit denen sich die Anwendung an der Datenbank anmelden kann. Damit alle Unicode-Zeichen korrekt verarbeitet werden, achten Sie bei der Anlage des Datenbankschemas darauf, die Sortierung entsprechend einzustellen.

Folgende Daten sind für die Konfiguration erforderlich:

- **Datenbanktyp**: Das verwendete Datenbankmanagementsystem, wie z.B. Microsoft SQL Server.
- **Hostname**: Der Name oder die IP-Adresse des Hosts, auf dem die Datenbank zu erreichen ist.
- **Port**: Der Port, unter dem die Datenbank auf dem angegebenen Host erreichbar ist.
- **Datenbankname**: Der Name der Datenbank.
- **Datenbankadministrator**: Der Name des Benutzers, der die Rechte besitzt, Schemaänderungen (z.B. das Anlegen von Tabellen oder Indizes) durchzuführen.
- **Passwort des Datenbankadministrators**: Das Passwort des Administrators
- **Datenbankbenutzer**: Der Name des Benutzers, der zur Laufzeit für den Zugriff auf die Datenbank verwendet werden soll.
- **Passwort des Datenbankbenutzers**: Das Passwort des Benutzers

Um die Datenbank zu konfigurieren, gehen Sie folgendermaßen vor.

So geht's

1. Wählen Sie das Feature **Konfiguration** aus.
2. Wählen Sie im Bereich **Prozesseinstellungen** den Eintrag **Datenbank** aus.
3. Tragen Sie die Verbindungsdaten ein.
4. Wählen Sie **Konfiguration speichern** aus.
5. Starten Sie die Anwendung neu.

> **Anmerkung**
>
> Für Microsoft SQL Server wird der JDBC-Treiber in d.velop process mitgeliefert.
>
> Wenn Sie eine andere Datenbank verwenden, benötigen Sie einen JDBC-Treiber, der den Systemvoraussetzungen entspricht. Mit dem entsprechenden JDBC-Treiber stellen Sie die Kommunikation mit dem Datenbankmanagementsystem sicher.

Verwenden einer Microsoft SQL Server-Datenbank

Beachten Sie bei der Verwendung und Konfiguration von Microsoft SQL Server die folgenden Hinweise:

- Um eine Verbindung zur Datenbank aufzubauen, aktivieren Sie das TCP/IP-Protokoll für den SQL Server.
- Die Tabellen werden im Default-Schema des administrativen Datenbanknutzers angelegt.
- Der administrative Benutzer muss für die Anlage von Tabellen und Indizes in seinem Default Schema berechtigt sein.
- Der Laufzeitbenutzer muss dasselbe Default-Schema verwenden wie der administrative Benutzer.
- Der Laufzeitbenutzer benötigt in diesem Schema die Berechtigungen **SELECT**, **INSERT**, **UPDATE** und **DELETE**.
- Konfigurieren Sie in der Datenbank für die beiden Schalter **ALLOW_SNAPSHOT_ISOLATION** und **READ_COMMITTED_SNAPSHOT** den Wert **ON** .

Verwenden einer Oracle-Datenbank

Beachten Sie bei der Verwendung und Konfiguration einer Oracle-Datenbank folgende Hinweise:

- Der administrative Benutzer benötigt die Berechtigungen **CREATE TABLE** und **CREATE SESSION**. Die Tabellen und Indizes werden im Schema und im Default Tablespace dieses Benutzers angelegt. Der Benutzer benötigt eine entsprechende Quota für den Tablespace.
- Um eine Verbindung zur Datenbank aufzubauen, wird ein Net Listener mit TCP-Protokoll benötigt. Tragen Sie den Port des Listeners bei **Port** ein.
- Tragen Sie bei **Datenbankname** den Service Name oder die SID ein. Dieser Name muss am Net Listener verfügbar sein.
- Die Benutzernamen sind case-sensitive.
- Der Laufzeitnutzer benötigt die Berechtigung **CREATE SESSION**.
- Der Laufzeitnutzer erhält die Berechtigungen **SELECT**, **INSERT**, **UPDATE** und **DELETE** für die Tabellen im Schema des administrativen Benutzers.

Ändern des Hostnames und Ports für die d.ecs http gateway-Registrierung (optional)

Für die Registrierung der Anwendung an d.ecs http gateway wird im Regelfall der lokale Computername und der im Setup konfigurierte Port verwendet.

Falls Sie diese Daten ändern möchten, können Sie die Änderungen in einer Konfigurationsdatei festlegen. Die Konfigurationsdatei finden Sie unter <Installationsverzeichnis der Anwendung>\conf\process-app.properties. Falls die Datei noch nicht existiert, können Sie eine Datei mit diesem Namen erstellen.

Sie können in der Datei die Eigenschaften **port** und **serverName** eintragen.

```
serverName=myhost.mydomain
port=8087
```

Nach einer Änderung müssen Sie die Anwendung neu starten, damit die Änderungen wirksam werden.

Einrichten von Benutzerrollen (optional)

Mit dem Eintrag **Benutzerrollen** im Feature **Konfiguration** können Sie Benutzern eine Benutzerrolle zuweisen. Jeder authentifizierte Benutzer ist nach der Einrichtung von d.velop process automatisch der Rolle Prozessbenutzer zugewiesen.

Sie können Benutzern folgende Rollen zuweisen:

- **Prozessadministrator**: Besitzt alle Berechtigungen für die Arbeit mit d.velop process.
- **Prozessbenutzer**: Darf Einmalprozesse starten und Prozesse bereitstellen.

Möchten Sie einem Benutzer beispielsweise die Rolle Prozessadministrator zuweisen, tragen Sie einfach den Benutzernamen in das entsprechende Feld ein und wählen ihn aus.

Einrichten von HTTPS (optional)

Sie können die Kommunikation zwischen d.ecs http gateway und der Anwendung mit HTTPS verschlüsseln.

So geht's

1. Erstellen Sie eine P12-Datei mit dem Namen **keystore.p12** im Konfigurationsordner des Installationsverzeichnisses der Anwendung (\conf\keystore.p12).
2. Erstellen Sie optional in demselben Verzeichnis die Datei **process-app.properties**, falls sie noch nicht vorhanden ist .
3. Fügen Sie der Datei folgende Zeilen hinzu:
   keystorePass=Passwort_des_Keystores
   keyAlias=Alias_des_Zertifikates (optional, wenn nur ein Alias im Keystore vorhanden ist, wird dieser automatisch genommen)
4. Starten Sie die Anwendung neu, um die Einrichtung abzuschließen.

Einrichten eines Clusters

Um mehrere Instanzen der Anwendung in einem Cluster zu installieren, müssen Sie sicherstellen, dass alle Instanzen die identische Konfiguration verwenden. Dazu muss auf jedem System, auf dem eine Installation der Anwendung vorgenommen werden soll, ein jstore mit Cluster-Konfiguration vorhanden sein. Dadurch registrieren sich alle Installationen am selben d.ecs http gateway.

Sie müssen außerdem sicherstellen, dass auch die Datenbankkonfiguration für alle Installationen identisch ist.

So geht's

1. Beenden Sie alle Instanzen der Anwendung bis auf eine. Dadurch stellen Sie sicher, dass genau eine Instanz konfiguriert wird.
2. Konfigurieren Sie die Datenbank für die Anwendung.
3. Starten Sie die nun konfigurierte Instanz neu, um die Änderungen wirksam zu machen.
4. Kopieren Sie im Installationsverzeichnis dieser Instanz die Dateien **conf\process-app-db.properties** und **lib\pa-jdbc-*.jar**.
5. Fügen Sie diese Dateien in das Installationsverzeichnis von allen anderen Installation der Anwendung ein.
6. Starten Sie nun alle Instanzen.

Dieses Verfahren können Sie auch anwenden, wenn Sie eine Änderung der Verbindungsdaten für die Datenbank vornehmen möchten.

Warnung

Vorgehen bei einem Software-Update:

Beachten Sie, dass vor dem Einspielen einer neuen Softwareversion alle Instanzen der Anwendung beendet werden müssen. Starten Sie anschließend nur die Instanzen neu, die bereits aktualisiert wurden.

Konfigurieren von Protokollexporten

Sie möchten den Export von Prozessprotokollen in ein d.3-Repository konfigurieren, um Protokolle langfristig zu speichern.

Das müssen Sie wissen

- Sie benötigen ein konfiguriertes d.3-Repository, in dem die Protokollexporte gespeichert werden.

- Sie müssen mit einem Benutzer angemeldet sein, der Zuordnungen für das d.3-Repository administrieren kann.
- Sie müssen in Ihrem Repository eine Dokumentart erstellen, die vier Eigenschaften vom Typ **A lphanumerisch** (je nach System auch **Text**) mit einer maximalen Länge von 250 Zeichen besitzt. Diese neuen Eigenschaften müssen Sie im Folgenden den Prozesseigenschaften zuordnen. Geben Sie den Eigenschaften sprechende Namen, die zu den Prozesseigenschaften passen, z.B. **Prozess**, **Prozessinstanz**, **Business-Key** und **Anhang**.
- Wenn einer der Werte länger als 250 Zeichen (oder 255 Bytes) ist, wird dieser Wert beim Export passend abgeschnitten. Die abgeschnittene Stelle wird mit drei Punkten gekennzeichnet.
- Wenn es beim Exportvorgang zu einer Kürzung von Metadaten oder einem Fehler kommt, sehen Sie einen entsprechenden Hinweis für die Prozessinstanz in der Prozessüberwachung sowie in der Protokollansicht.
- Sie müssen einen API-Schlüssel für einen Benutzer erstellen, der Administrationsrechte besitzt. Darüber hinaus benötigt der Benutzer lesenden und schreibenden Zugriff für diese Dokumentart.

So geht's

1. Öffnen Sie das Feature **Zuordnungen**.
2. Fügen Sie eine neue Zuordnung hinzu.
3. Wählen Sie die Quelle **Prozesse** aus.
4. Fügen Sie eine Kategorie hinzu und verknüpfen Sie die Quelle **Prozessprotokoll** mit der gewünschten Dokumentart.
5. Verknüpfen Sie alle Eigenschaften der Quelle mit Dokumenteigenschaften des Ziels. Stellen Sie sicher, dass alle vier Eigenschaften zugeordnet sind.
6. Speichern Sie die Zuordnung.
7. Öffnen Sie das Feature **Konfiguration** und wählen Sie **DMS-Exportkonfiguration** im Bereich **Prozesseinstellungen** aus.
8. Wählen Sie das Zielrepository aus und tragen Sie den API-Schlüssel ein.
9. Speichern Sie die Änderungen.

Nachdem Sie den Protokollexport konfiguriert haben, können Sie die Option zum Exportieren des Protokolls beim Bereitstellen einer Prozessdefinition aktivieren.

Ändern der Basisadresse des Clients (optional)

d.velop process erzeugt z.B. bei Benachrichtigungen, URLs, die von einem Client (z.B. Webbrowser) geöffnet werden können. Die URLs beginnen mit der Basisadresse des Systems.

Sie können bei Bedarf eine alternative Basisadresse für die URL in einer Konfigurationsdatei festlegen. Sie finden die Datei unter <Installationsverzeichnis der Anwendung>\conf\process-app.properties\.

Tragen Sie die Eigenschaften in der Datei **clientBaseUri** ein.

```
clientBaseUri=https://myhost.mydomain
```

Starten Sie anschließend die Anwendung neu, damit die Änderungen wirksam werden.

1.3.2. Konfigurieren von d.velop process modeler

In diesem Thema finden Sie Informationen zur Konfiguration von d.velop process modeler.

Einrichten von Benutzerrollen für d.velop process modeler

Für d.velop process modeler müssen Sie keine separaten Benutzerrollen konfigurieren. An dieser Stelle wird auf die Konfiguration von d.velop process zurückgegriffen. Um Prozesse modellieren zu können, müssen Sie in d.velop process als Administrator eingetragen sein.

Siehe auch: Einrichten von Benutzerrollen (optional)

Einrichten von HTTPS für d.velop process modeler (optional)

Sie können die Kommunikation zwischen d.ecs http gateway und der Anwendung mit HTTPS verschlüsseln.

So geht's

1. Erstellen Sie eine Datei **keystore.p12** im Konfigurationsordner des Installationsverzeichnisses der Anwendung (**\config**).
2. Kopieren Sie die Datei **application.properties.template** im Verzeichnis **\config** und benennen Sie die kopierte Datei in **application.properties** um.
3. Tragen Sie in der Datei die folgenden Konfigurationsparameter ein und aktivieren Sie sie:
   a. **quarkus.http.ssl.certificate.key-store-file=Pfad_zur_Keystore-Datei**
   b. **quarkus.http.ssl.certificate.key-store-password=Passwort_des_Keystores**
4. Fügen Sie in der Datei optional den folgenden Konfigurationsparameter hinzu und aktivieren ihn: **quarkus.http.ssl.certificate.key-store-key-alias=Alias_des_Zertifikates** (optional; wenn nur ein Alias im Keystore vorhanden ist, wird dieser automatisch genommen)
5. Starten Sie die Anwendung neu, um die Einrichtung abzuschließen.

1.3.3. Konfigurieren von d.velop forms

In diesem Thema finden Sie Informationen zur Konfiguration von d.velop forms

Einrichten von Benutzerrollen für d.velop forms

Im Feature **Konfiguration** können Sie im Bereich **Formulare > Benutzerrollen** Benutzern oder Gruppen eine Benutzerrolle zuweisen. Standardmäßig können alle Benutzer Formulare öffnen, ausfüllen und absenden, auch wenn sie keiner Rolle zugewiesen sind. Mit Benutzerrollen werden bestimmten Benutzern zusätzliche Berechtigungen bei der Arbeit mit Formularen zugewiesen.

Folgende Rollen sind verfügbar:

- **Administration**:
  Der Benutzer bzw. die Gruppe kann Formulare erstellen, bearbeiten und löschen.
- **Bearbeitung:**
  Der Benutzer bzw. die Gruppe kann Formulare erstellen.
  Der Benutzer bzw. die Gruppe kann Formulare bearbeiten und löschen, sofern der Benutzer bzw. die Gruppe explizite Berechtigungen für diese Formulare hat.

Um einem Benutzer oder einer Gruppe eine Rolle zuzuweisen, tragen Sie den Namen des Benutzers bzw. der Gruppe in das entsprechende Feld ein und wählen Sie den Benutzer bzw. die Gruppe aus.

1.3.4. Konfigurieren von d.velop scripting

In diesem Thema finden Sie Informationen zur Konfiguration von d.velop scripting.

Einrichten von Benutzerrollen für d.velop scripting

Im Feature **Konfiguration** können Sie im Bereich **Skripting > Benutzerrollen** Benutzern oder Gruppen eine Benutzerrolle zuweisen. Standardmäßig können alle Benutzer Skripte ausführen, auch wenn sie keiner Rolle zugewiesen sind. Mit Benutzerrollen werden bestimmten Benutzern zusätzliche Berechtigungen bei der Arbeit mit Skripten zugewiesen.

Folgende Rollen sind verfügbar:

- **Administration**:
  Der Benutzer bzw. die Gruppe kann Skripte erstellen, bearbeiten und löschen.
- **Bearbeitung:**
  Der Benutzer bzw. die Gruppe kann Skripte erstellen.
  Der Benutzer bzw. die Gruppe kann Skripte bearbeiten und löschen, sofern der Benutzer bzw. die Gruppe explizite Berechtigungen für diese Skripte hat.

Um einem Benutzer oder einer Gruppe eine Rolle zuzuweisen, tragen Sie den Namen des Benutzers bzw. der Gruppe in das entsprechende Feld ein und wählen Sie den Benutzer bzw. die Gruppe aus.

Verwenden von d.velop scripting in Verbindung mit einem Proxy-Server

Sie können d.velop scripting in Verbindung mit einem Proxy-Server verwenden.

So geht's

1. Erstellen Sie eine Systemumgebungsvariable mit der Bezeichnung **DVS_HTTP_PROXY**.
2. Tragen Sie in der Systemumgebungsvariable den entsprechenden Proxy-Server ein. Beachten Sie folgende Hinweise:
   - Der Eintrag muss mit **http://** oder **https://** beginnen.
   - Wenn der Proxy-Server nur ein selbstsigniertes Zertifikat anbietet, müssen Sie eine weitere Systemumgebungsvariable erstellen. Geben Sie der Systemumgebungsvariable die Bezeichnung **DVS_HTTP_PROXY_UNSAFE_ALLOW_SELF_SIGNED_CERTIFICATES** und den Wert **true**.
3. Erstellen Sie die Node.js-Konfiguration des ausführenden d.velop scripting-Prozessbenutzers (Servicebenutzer von d.3 process manager). Geben Sie hierzu im Kontext des Prozessbenutzers den entsprechenden Befehl ein:
   - Bei Verwendung eines HTTP-Proxy-Servers: **npm config set proxy http://proxy_host:port npm config set https-proxy http://proxy.company.com:8080**
   - Bei Verwendung eines HTTPS-Proxy-Servers: **npm config set https-proxy https://proxy.company.com:8080 npm config set strict-ssl false**

Betreiben von d.velop scripting im Cluster

Wenn Sie d.velop scripting auf mehreren Servern im Cluster betreiben, muss die Datei **top.secret** im Arbeitsverzeichnis aller Server identisch sein. Kopieren Sie die Datei nach dem ersten Start von d.velop scripting auf alle Knoten, um die verschlüsselten Variablen der Skripte auf allen Knoten entschlüsseln zu können.

Überschreiben des FQDN für die Registrierung an d.ecs http gateway (optional)

d.velop scripting bestimmt den FQDN (Fully-Qualified Domain Name) für die Registrierung an d.ecs http gateway automatisiert. In einigen Fällen versucht die App sich mit einem FQDN zu registrieren, der von d.ecs http gateway nicht erreicht werden kann.

Sie können den FQDN für die Registrierung mithilfe einer Umgebungsvariable von Windows übersteuern.

Erstellen Sie eine Systemumgebungsvariable mit dem Namen **DVS_FQDN** und tragen Sie dort den zu verwendenden FQDN in der Form **test.meinedomaene.local** ein.

> **Anmerkung**
> Nachdem Sie die Umgebungsvariable erstellt oder geändert haben, muss d.3 process manager gestartet werden. Dadurch werden die Änderung von d.velop scripting übernommen.

1.3.5. Konfigurieren von d.velop actions eventbridge

In diesem Thema finden Sie Informationen zur Konfiguration von d.velop actions eventbridge.

Einrichten eines Clusters

Wenn Sie mehrere Instanzen der Anwendung in einem Cluster installieren möchten, müssen Sie sicherstellen, dass alle Instanzen die identische Konfiguration verwenden. Auf jedem System mit einer Installation muss d.ecs jstore mit einer Cluster-Konfiguration vorhanden sein. Dadurch registrieren sich alle Installationen an derselben d.ecs http gateway-Instanz.

Zusätzlich müssen auch die verschlüsselten Daten und optional die HTTPS-Konfiguration auf allen Instanzen identisch sein.

So geht's

1. Kopieren Sie (sofern vorhanden) die Dateien **.env**, **eventbridge.crt**, **eventbridge.key** sowie **top.secret** aus dem Installationsverzeichnis der primären Instanz.
2. Fügen Sie die kopierten Dateien im Installationsverzeichnis aller übrigen Instanzen ein.
3. Starten Sie die Instanzen neu.

So gehen Sie bei einem Softwareupdate vor

Beenden Sie vor dem Einspielen einer neuen Softwareversion alle Instanzen der Anwendung. Starten Sie anschließend nur die Instanzen neu, die bereits aktualisiert wurden.

Einrichten von HTTPS (optional)

Sie können die Kommunikation zwischen d.ecs http gateway und der Anwendung mit HTTPS verschlüsseln.

So geht's

1. Erstellen Sie eine öffentliche Zertifikatsdatei mit dem Namen **eventbridge.crt** im Installationsverzeichnis der Anwendung.
2. Erstellen Sie eine Datei mit dem privaten Schlüssel des Zertifikats mit dem Namen **eventbridge.key** im Installationsverzeichnis der Anwendung.
3. Erstellen Sie im Installationsverzeichnis die Datei **.env** (sofern noch nicht vorhanden).
4. Fügen Sie der Datei folgende Zeilen hinzu:

```
https.enabled=true
https.certificate.public.path=./eventbridge.crt
https.certificate.private.path=./eventbridge.key
```

5. Starten Sie die Anwendung neu, um die Einrichtung abzuschließen.

Beachten Sie auch die Hinweise im Kapitel Einrichten eines Clusters.

1.4. Modellieren von Prozessen

In diesem Thema erfahren Sie, wie Sie Ihre eigenen Prozesse modellieren. Sie lernen die unterstützten BPMN-Elemente kennen und erfahren, wie Sie diese zu Ihren Prozessen hinzufügen.

1.4.1. Wissenswertes zur Prozessmodellierung

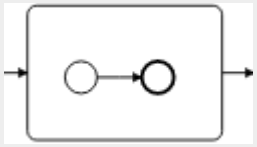
Bei der Erstellung von Prozessmodellen unterstützt die Anwendung den BPMN-Standard. Dabei handelt es sich um ein XML-basiertes Format. Das Modellieren von Prozessen im d.velop-Kontext orientiert sich am Standard der Object Management Group (OMG). Der Wert **http://www.omg.org/spec/BPMN/20100524/MODEL** ist der standardmäßige Namensraum.

Für die Modellierung von Prozessmodellen empfehlen wir die Verwendung von d.velop process modeler.

1.4.2. Verwenden von BPMN-Elementen

Bei der Erstellung eigener Prozessmodelle können Sie folgende BPMN-Elemente des BPMN-Standards 2.0 verwenden:

	BPMN-Element	Restriktionen
	**Start event** (Startereignis)	Erlaubte Ereignistypen: None, Timer Nur in eventbasierten Subprozessen: Eskalation, Error

	BPMN-Element	Restriktionen
	**End event** (Endereignis)	Erlaubte Ereignistypen: Keine, Terminierung, Eskalation, Error.
  	**Intermediate boundary event** (angeheftetes Zwischenereignis)	Erlaubte Ereignistypen: Timer, Eskalation, Error.
	**Intermediate catch event** (Zwischenereignis mit wartender Funktion)	Erlaubte Ereignistypen: Timer.
	**Intermediate throw event** (Zwischenereignis mit auslösender Funktion)	Erlaubte Ereignistypen: Eskalation.
	**User Task** (Benutzeraktivität)	
	**Send Task** (Sendenaktivität)	Nur zur Anbindung von Services.
	**Receive Task** (Empfangenaktivität)	Nur zur Anbindung von Services.
	**Service Task** (Serviceaktivität)	Nur definierte Aktionen (**writeLocalVariables**, **writeGlobalVariable** oder **writerMultiInstanceVariables**) erlaubt.
  	**Gateway** (Verzweigung)	Erlaubte Verzweigungstypen: Exclusiv, Parallel, Inclusiv.
	**Sequence flow** (Sequenzfluss)	
	**Sub process** (Subprozesse)	Erlaubte Subprozesstypen: Normal, eventbasiert (s. Startereignis)
	**Pool/Participant/Lane**	Pro Prozessdiagram ist maximal ein Pool bzw. Participant erlaubt.

Allgemeine Einschränkungen

Für alle Elemente gelten folgende Restriktionen:

- Ausdrücke bei In- und Output-Parametern unterliegen den Restriktionen für Ausdrücke.
- Es sind ausschließlich die folgenden camunda-Erweiterungen erlaubt:
  - **camunda:inputOutput**
  - **camunda:properties**
  - **camunda:property**
  - **camunda:failedJobRetryTimeCycle**
  - **camunda:asyncBefore**
  - **camunda:asyncAfter**
  - **camunda:formKey**

1.4.3. Erstellen eines exemplarischen Prozesses in einer BPMN-Datei

Für den Einstieg in das Modellieren eigener Prozesse ist es hilfreich, wenn Sie zunächst einen einfachen Prozess beispielhaft erstellen. Voraussetzung für das Modellieren eigener Prozesse sind eingerichtete Benutzerrollen. Außerdem benötigen Sie einen Text- oder BPMN-Editor mit XML-Ansicht und ein Tool zum Aufrufen einer REST-API.

Ein einfacher Prozess sieht beispielsweise so aus:

Erstellen Sie zum Modellieren Ihres eigenen Prozesses zunächst folgendes Grundgerüst in Ihrer BPMN-Datei:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process>
    <startEvent />
    <sequenceFlow />
    <userTask />
    <sequenceFlow />
    <endEvent />
  </process>
</definitions>
```

Diesem Grundgerüst fügen Sie die einzelnen Prozesselemente hinzu.

Hinzufügen des Prozesselements

Das BPMN-Element **process** ist eine Kapsel um den gesamten Prozess. Fügen Sie dem BPMN-Element die Eigenschaften **id**, **name** und **isExcecutable** hinzu.

```
<process id="hello_world_process" name="Hello World Process"
isExecutable="true">
```

Erläuterung der Eigenschaften

- **id**: Die Eigenschaft wird als eindeutiger Bezeichner für den Prozess verwendet.
- **name**: Die Eigenschaft wird als Anzeigename in den Benutzeroberflächen verwendet.
- **isExecutable**: Die Eigenschaft muss den Wert **true** enthalten, damit der Prozess gestartet werden kann.

Hinzufügen eines Start- und Endereignisses

Die BPMN-Elemente **startEvent** und **endEvent** werden als Markierungen verwendet, die anzeigen, wo der Prozess beginnt und wo er endet. Fügen Sie den BPMN-Elementen jeweils die Eigenschaft **id** hinzu.

```
<startEvent id="start" />
<endEvent id="end" />
```

Hinzufügen einer Benutzeraktivität

Das BPMN-Element **userTask** stellt eine Benutzeraktivität im Prozess dar. Bei Eingang in diese Aktivität wird einem oder mehreren Benutzern eine Aufgabe zugestellt. Fügen Sie dem BPMN-Element die Eigenschaften **id**, **name** und **camunda:candidateUsers** hinzu.

```
<userTask id="task_hello_world" name="Hello World" camunda:candidateUsers="${variables.get('dv_initiator')}" />
```

Erläuterung der Eigenschaften

- **id**: Die Eigenschaft dient als Bezeichner der Aktivität im BPMN-Modell und wird bei der Verbindung durch Sequenzflüsse verwendet.
- **name**: Die Eigenschaft wird der Aufgabe als Betreff hinzugefügt.
- **camunda:candidateUsers**: Die Eigenschaft bezieht sich auf die Person, die die Aufgabe bearbeiten soll. In der BPMN-Datei wird mit dem Ausdruck **${variables.get("dv_initiator")}** eine Variable verwendet. Die Variable **dv_initiator** verweist immer auf die Person, die den Prozess gestartet hat. Diese Aufgabe wird also immer der Person zugestellt, die den Prozess startet.

Hinzufügen von Sequenzflüssen

Verbinden Sie die einzelnen Prozesselemente, um einen Ablauf zu erstellen. Zum Verbinden der Prozesselemente verwenden Sie das BPMN-Element **sequenceFlow**. Fügen Sie dem BPMN-Element die Eigenschaften **id**, **sourceRef** und **targetRef** hinzu.

Die beispielhafte Konfiguration in der BPMN-Datei zeigt, wie Sie das Startereignis mit der Benutzeraktivität verbinden und die Benutzeraktivität mit dem Endereignis verknüpfen:

```
<sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
<sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
```

Erläuterung der Eigenschaften

- **id**: Die Eigenschaft ist ein eindeutiger Bezeichner für den Sequenzfluss innerhalb des BPMN-Modells.
- **sourceRef**: Die Eigenschaft verweist auf den Ausgangspunkt des Sequenzflusses.
- **targetRef**: Die Eigenschaft verweist auf das Ziel des Sequenzflusses. In dieser Eigenschaft tragen Sie die Bezeichner der zu verbindenden Elemente ein.

Nachdem Sie die einzelnen Prozesselemente hinzugefügt haben, sieht die BPMN-Datei folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="hello_world_process" name="Hello World Process"
```

```
isExecutable="true">
    <startEvent id="start" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world"
/>
        <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
    <endEvent id="end" />
  </process>
</definitions>
```

Hinzufügen von grafischen Informationen

Wenn Sie Ihren erstellten Prozess in einem Diagramm anzeigen möchten, können Sie der BPMN-Datei grafische Informationen hinzufügen. Diese grafischen Informationen legen die Größe des Diagramms und die Position der einzelnen Elemente fest. Die grafischen Informationen fügen Sie mithilfe des BPMN-Elements **bmpndi:BPMNDiagram** hinzu.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:di="http://www.omg.org/spec/DD/
20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
targetNamespace="" exporter="d.velop process modeler">
  <process id="hello_world_process" name="Hello World Process"
isExecutable="true">
    <startEvent id="start" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
    <endEvent id="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane" bpmnElement="hello_world_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="173" y="102" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="209" y="120" />
        <di:waypoint x="259" y="120" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="259" y="80" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="359" y="120" />
        <di:waypoint x="409" y="120" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="409" y="102" width="36" height="36" />
      </bpmndi:BPMNShape>
    </bpmndi:BPMNPlane>
```

```
  </bpmndi:BPMNDiagram>
</definitions>
```

1.4.4. Verwenden von Ausdrücken

Sie können beim Modellieren von Prozessen variable Ausdrücke verwenden. Die Ausdrücke verwenden die Syntax der Java Expression Language.

Ausdrücke dürfen lediglich folgende Elemente verwenden:

Element	Restriktion
Lesender Zugriff auf die Variablen mittels **variables** • **variables.get** • **variables.getDisplayValue** • **variables.getObjectValue**	
Erzeugen von Mehrfachwerten • **collections.from**	

Lesender Zugriff auf Werte der aktuellen Prozessinstanz
 	
• **process.instance.id**
  • ID der aktuellen Prozessinstanz
• **process.instance.businessKey**
  • Business-Key der aktuellen Prozessinstanz	

Operatoren und Zeichen	Erlaubte Operatoren und Zeichen: +, -, *, /, %, =, !, <, >, &, |, ?, (,), [,].
Keyword	Erlaubte Keywords: div, mod, eq, ne, lt, gt, le, ge, and, or, not, empty.
Numerische Werte	
Zeichenketten	Zeichenketten müssen in " " oder ' ' eingeschlossen sein.
Kürzen von Zeichenketten: • **strings.shorten**	

Beispiele

```
${"Hello World"}                                                    //
Hello World
${3}                                                                // 3
${3 > 4}                                                            //
false
${variables.get("myVariable")}                                      //
Value of myVariable
${variables.get("myVariable") == "Hello"}                           //
true, if myVariable equals Hello, false otherwise
${collections.from("a", "b", "c")}                                  //
creates a collection with "a", "b" and "c"
${strings.shorten("${variables.get('myLongTextVariable')}", 10)}    //
creates a substring of myVariable with the length of 10
```

Es sind auch Mischformen möglich, sodass Sie Texte und variable Inhalte direkt miteinander kombinieren können.

Beispiele

```
This is an expression with ${variables.get("myVariable")}
3 is ${(3 > 4) ? "greater" : "lesser"} than 4
```

1.4.5. Arbeiten mit Prozessvariablen

In diesem Thema erfahren Sie, wie Sie Ihre Prozesse mithilfe von Variablen dynamisch erstellen.

Wissenswertes zu Prozessvariablen

Prozessvariablen sind Daten, die Informationen zu einer Prozessinstanz enthalten. Folgende Prozessvariablen sind in jeder Prozessinstanz vorhanden:

- **dv_initiator**: Eine Prozessvariable vom Datentyp **Identity**. Die Prozessvariable verweist auf die Identität des Benutzers, der die Prozessinstanz gestartet hat.
- **dv_start_time**: Eine Prozessvariable vom Datentyp **String**. Die Prozessvariable enthält den UTC-String des Zeitpunktes, in dem die Prozessinstanz aus Sicht der Anwendung gestartet wurde. Es gilt der Zeitpunkt, in dem die Anwendung beauftragt wurde, die Prozessinstanz zu starten. Da der eigentliche (technische) Start der Instanz asynchron durchgeführt wird, kann der in der Variable enthaltene Zeitpunkt möglicherweise abweichen.

Verwenden weiterer Prozessvariablen

Diese Variablen können Sie für spezielle Zwecke verwenden:

- **dv_sender**: Eine Prozessvariable vom Datentyp **Identity**. Die Prozessvariable verweist auf die Identität des Benutzers, der als Absender von Benutzeraktivitäten verwendet wird.
- **dv_attachment**: Eine Prozessvariable vom Datentyp **DmsObject** oder abweichend. Die Prozessvariable enthält eine Verknüpfung, die allen Benutzeraktivitäten als Anhang hinzugefügt wird.
- **dv_decision:** Eine Prozessvariable vom Typ **String**. Dieser Text wird im Prozessprotokoll bei einer Aufgabe als **Entscheidung** hervorgehoben.
- **dv_comment:** Eine Prozessvariable vom Typ **String**. Dieser Text wird im Prozessprotokoll bei einer Aufgabe als **Kommentar** hervorgehoben.

> ### Warnung
> Die Prozessvariable **dv_attachment** ist mit dem Datentyp **DmsObject** vordefiniert. Möchten Sie eine andere URI als die zu einem Objekt der DMS-App verwenden, müssen Sie den Wert des Datentyps auf **String** ändern.

Definieren von Prozessvariablen

Definieren Sie in der Prozessdefinition Variablen, die innerhalb eines Prozesses genutzt werden. Definierte Variablen können an den entsprechenden Schnittstellen der Anwendung automatisch validiert und ggf. konvertiert werden.

Innerhalb eines Prozesses können Sie folgende Variablen definieren:

- **Allgemeine Prozessvariable**: Eine Variable, die im gesamten Prozess Gültigkeit hat. Sie können festlegen, ob es sich um eine Startvariable handelt.
- **Lokale Prozessvariable**: Eine Variable, deren Gültigkeitsbereich auf eine einzelne Prozessaktivität beschränkt ist.
- **Servicevariablen**: Ein- oder Ausgabevariablen eines Prozessservices.

Eine Variablendefinition beinhaltet folgende Informationen:

Eigenschaft	Mögliche Werte	Beispiel
Name	[A-Za-z][A-Za-z0-9_]*	myVariable (**name**-Eigenschaft)
Startvariable		myStartVariable* (**name**-Eigenschaft)

Eigenschaft	Mögliche Werte	Beispiel
Datentyp	• String • Number • Boolean • Identity • DmsObject • URL • Object • File	String (**value**-Eigenschaft)
Einzel- oder Mehrfachwert		Einzelwert: String (**value**-Eigenschaft) Mehrfachwert: [String] (**value**-Eigenschaft)
Pflichtvariable		Einzelwert: String! (**value**-Eigenschaft) Mehrfachwert: [String]! (**value**-Eigenschaft)

Anmerkung

Besonderheiten des Datentyps **Object**:

- Sie können in einem Prozess nur eine Variable vom Typ **Object** definieren.
- Eine Variable vom Typ **Object** darf keinen Mehrfachwert enthalten.
- Variablen vom Typ **Object** werden nicht protokolliert.

Anmerkung

Besonderheiten des Datentyps **File**:

- Eine Datei darf maximal 100 MB groß sein.
- Die Gesamtgröße aller Dateien einer Prozessinstanz darf 500 MB nicht überschreiten.
- Dateien werden nach Prozessende gelöscht.
- Variablen vom Typ **File** werden nicht protokolliert.

Warnung

Beachten Sie beim Verwenden von Prozessvariablen folgende wichtige Hinweise:

- **Namenspräfix "dv_"**: Sie dürfen für Variablen das Namenspräfix **dv_** nicht verwenden. Diese Variablen sind für interne Variablen reserviert. Sie dürfen das Namenspräfix **dv_** lediglich beim Überschreiben der Variablendefinition **dv_attachment** verwenden.
- **Ändern von Variablendefinitionen beim Versionswechsel**: Wenn Sie zwischen verschiedenen Versionen eines Prozesses den Datentypen einer Variablen ändern oder zwischen Einzelwert und Mehrfachwert wechseln, kann es nach der Migration zu Fehlern in der Prozessausführung kommen. Die Fehler treten auf, wenn bestehende Prozessinstanzen für die entsprechende Variable bereits Werte der bisherigen Definition beinhalten. Nach einer Migration in die neue Version verlieren diese Werte ihre Gültigkeit. Stellen Sie bei Änderungen der Variablendefinitionen immer sicher, dass keine zu migrierende Prozessinstanz betroffen ist.

Anmerkung

Sie können eine Variable, die als Pflichtvariable definiert ist, mit der Schnittstelle zum Ändern von Variablen nicht löschen. Handelt es sich bei der Pflichtvariable zusätzlich um eine Startvariable, muss ein Anwender beim Start einer Prozessinstanz immer einen gültigen Wert für die Variable übergeben. Wenn er keinen gültigen Wert übergibt, wird die Anfrage mit dem Fehlercode 400 abgelehnt.

Für den Datentypen **Number** gelten diese Einschränkungen:

Minimum	Maximum	Nachkommastellen	Genauigkeit
-1e16	1e16	5	15 Stellen

Die nicht-primitiven Datentypen verwenden die folgende Notation:

Datentyp	Notation
Identity	**Für Benutzer:** identity:///identityprovider/scim/users/<someUserId> **Für Gruppen:** identity:///identityprovider/scim/groups/<someGroupId>
DmsObject	dmsObject:///dms/r/<repositoryId>/o2/<dmsObjectId>
URL	https://www.d-velop.de **(max. 2000 Zeichen)**
Object	{ "myKey" : "myValue"} **(JSON-String, max. 50 KB)**
File	Variablen vom Typ **File** können im BPMN nicht gesetzt werden. Dies ist nur über die API möglich.

Variablen werden als BPMN-Erweiterung in Form von Camunda-Properties definiert. Allgemeine Prozess- und Servicevariablen definieren Sie innerhalb des BPMN-Modells direkt im Prozessknoten. Lokale Prozessvariablen definieren Sie im Knoten der Aktivität, in der sie gültig sein sollen.

Allgemeine Prozess- und Servicevariablen

```
 <process id="myProcess" name="Process with variable declaration"
isExecutable="true">
    <extensionElements>
        <camunda:properties>
            <camunda:property name="variable:myVariable" value="[String]!"
/> <!-- Name: "myVariable", Type: "String", Multiple: true, Mandatory:
true, Startvariable: false -->
            <camunda:property name="variable:myStartVariable*"
value="String" /> <!-- Name: "myStartVariable", Type: "String", Multiple:
false, Mandatory: false, Startvariable: true-->
            <camunda:property name="variable:someUserTaskOutput"
value="String" /> <!-- Name: "someUserTaskOutput", Type: "String",
Multiple: false, Mandatory: false, Startvariable: false -->
            <camunda:property name="service:/myservice:in:input"
value="String!" /> <!-- Name: "input", Type: "String", Multiple: false,
Mandatory: true -->
            <camunda:property name="service:/myservice:out:output"
value="Number!" /> <!-- Name: "output", Type: "Number", Multiple: false,
Mandatory: true -->
        </camunda:properties>
    </extensionElements>
</userTask>
```

Lokale Prozessvariable

```
<userTask id="userTask" name="User Task">
    <extensionElements>
        <camunda:inputOutput>
            <camunda:inputParameter name="someInput">user task input which
is only available within this user task scope</camunda:inputParameter>
            <camunda:outputParameter name="someUserTaskOutput">user task
output which will be written to process scope</camunda:outputParameter>
        </camunda:inputOutput>
        <camunda:properties>
            <camunda:property name="variable:someInput" value="String" />
```

```
        </camunda:properties>
    </extensionElements>
</userTask>
```

Verwenden von Prozessvariablen

Beim Modellieren eigener Prozesse können Sie Prozessvariablen verwenden, um dynamische Prozesse zu erstellen. Zum Einstieg in das Arbeiten mit Prozessvariablen ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Dieser Prozess sieht beispielsweise so aus:

Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut. Zur Vereinfachung sind die grafischen Informationen zum BPMN-Diagramm in dieser BPMN-Beispieldatei nicht enthalten.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="GettingStarted" name="Beispiel: Einstieg"
isExecutable="true">
    <startEvent id="start"/>
    <userTask id='task_hello_world' name='Hello World'
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
        <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end"
/>
    <endEvent id="end"/>
  </process>
</definitions>
```

Dieser Prozessdefinition fügen Sie nun eine neue Variablendefinition hinzu.

Hinzufügen einer Variablendefinition

Um eine Prozessvariable verwenden zu können, müssen Sie diese der Prozessdefinition hinzufügen. In der Prozessdefinition fügen Sie die BPMN-Elemente **extensionElements** und **camunda:properties** zum BPMN-Element **process** hinzu. Zum Definieren der Prozessvariable fügen Sie dem BPMN-Element **camunda:properties** noch das Element **camunda:property** hinzu.

Dieses Beispiel zeigt, wie Sie die Variable **message** vom Datentyp **String** hinzufügen:

```
<process id="GettingStarted" name="Beispiel: Einstieg" isExecutable="true">
        <extensionElements>
      <camunda:properties>
        <camunda:property name="variable:message" value="String" />
      </camunda:properties>
    </extensionElements>
        ...
</process>
```

Erläuterung der Eigenschaften

- **name**: Die Eigenschaft definiert den Namen der Prozessvariable. Vor dem Wert steht immer das Präfix **variable:**.
- **value**: Die Eigenschaft definiert den Datentypen der Prozessvariable.

Verwenden der Prozessvariable

Sie können die Variable **message** nun für den Namen der Benutzeraktivität verwenden, damit dieser Name dem Anwender in der Aufgabenliste angezeigt wird. Hierzu ersetzen Sie den vorhandenen Text in der Eigenschaft **name** der Benutzeraktivität durch die Methode für den Zugriff auf die Variable **message**:

```
<process id="GettingStarted" name="Beispiel: Einstieg" isExecutable="true">
        ...
        <userTask id="task_hello_world" name="${variables.get('message')}"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
        ...
```

Sie können den Wert der Variablen nun zum Beispiel beim Start des Prozesses belegen.

Ermitteln von Prozessvariablen

Mit dem Objekt **variables** können Sie auf die Prozessvariablen zugreifen. Der Aufruf erfolgt jeweils in der Notation **variables.<methodName>(<Parameter>)**.

Sie können die Methoden nicht auf Prozessvariablen vom Typ **File** anwenden.

get(String variableName)

Die Methode ermittelt den Wert für die Variable mit dem eingegebenen Variablennamen. Bei Variablen mit einem Mehrfachwert werden die Werte in einer Liste (collection) zurückgegeben.

getDisplayValue(String variableName)

Die Methode ermittelt den Wert für die Variable mit dem eingegebenen Variablennamen. Je nach Datentyp findet hierbei automatisch eine entsprechende Konvertierung statt.

Bei Variablen mit einem Mehrfachwert werden die konvertierten Werte in einem String mit Kommas getrennt zurückgegeben. Sind die Variablen vom Datentyp **Number**, werden die konvertierten Werte in einem String mit Semikolons getrennt zurückgegeben.

Datentyp	Tatsächlicher Wert	Rückgabewert
String	"some value"	"some value"
Number	1.23	"1.23"
Boolean	true	"true"
Identity	identity:///identityprovider/scim/users/userIdOf-MaxMustermann	"Max Mustermann" (**displayName**-Eigenschaft des Benutzers, **name**-Eigenschaft bei Gruppen im d.ecs identity provider)
DmsObject	dmsObject:///dms/r/someRepo/o2/T000000001	"/dms/r/someRepo/o2/T000000001"
Object	{"myKey":"myValue"}	"{\"myKey\":\"myValue\"}" (JSON-String)

getObjectValue(String variableName, String objectPath)

Diese Methode kann nur für Variablen vom Typ **Object** verwendet werden. Sie ermöglicht es, auf einzelne Teile des Objekts zuzugreifen.

Beispiel:

Angenommen, es gibt eine Variable vom Typ **Object** mit dem Namen "myObjectVariable" und folgendem Wert:

```
{
  "my" : {
    "object" : {
      "path" : 123
    }
  }
}
```

Dann können mit der Methode **getObjectValue** z. B. die folgenden Teile des Objekts ermittelt werden:

Methodenaufruf	Rückgabewert
getObjectValue("myObjectVariable", "my.object.path")	123
getObjectValue("myObjectVariable", "my.object")	{"path":123}

Erzeugen von Mehrfachwerten

Mit dem Objekt **collections** können Sie aus Einzelwerten eine Liste (collection) erzeugen, z.B. um einer Variablen einen Mehrfachwert zuzuweisen. Das Erzeugen einer Liste aus Einzelwerten ist beispielsweise hilfreich, wenn Sie in einer Benutzeraktivität mehrere Empfänger als Konstante hinterlegen wollen.

Beispiel

```
${collections.from( "identity:///identityprovider/scim/users/user1",
"identity:///identityprovider/scim/users/user")}
```

from(Object...objects)

Die Methode erzeugt eine Liste (collection) mit den übergebenen Einzelwerten.

Wissenswertes zum Geltungsbereich von Prozessvariablen

Prozessvariablen haben innerhalb einer Prozessinstanz immer einen bestimmten Geltungsbereich. Je nach Geltungsbereich sind lesende und schreibende Zugriffe möglich.

In jeder Prozessinstanz gibt es einen universellen Geltungsbereich, der den gesamten Prozess umfasst. Alle weiteren Geltungsbereiche, wie zum Beispiel Benutzer- oder Serviceaktivitäten, können lesend auf die Variablen im universellen Geltungsbereich zugreifen.

Zusätzliche Geltungsbereiche werden an folgenden Stellen erzeugt:

- Verzweigungen mit parallelen Pfaden
- Multi-Instanz-Konstrukte
- Subprozesse
- Input-/Output-Mapping oder Timer-Ereignis bei Benutzeraktivitäten
- Serviceaktivitäten

Lesende Zugriffe

Lesende Zugriffe auf Prozessvariablen (z.B. die Verwendung in einem Ausdruck) sind unabhängig vom Geltungsbereich möglich. Wird im aktuellen Geltungsbereich kein Wert gefunden, so wird der darüber liegende Bereich durchsucht, bis ein Wert gefunden wurde.

Schreibende Zugriffe

Wenn Sie Werte ändern möchten (z.B. in einer Benutzeraktivität mit Output-Mapping), so wird der Wert immer zunächst in den aktuellen Geltungsbereich der Aktivität übernommen. Andere Geltungsbereiche, z.B. entlang anderer Zweige im Prozess, werden dadurch bis zum Ende dieses Geltungsbereiches nicht beeinflusst. Erst am Ende des aktuellen Wertebereichs werden die Änderungen auch in alle anderen Geltungsbereiche übernommen.

Ende von Geltungsbereichen

Am Ende eines Geltungsbereichs (z.B. bei der Zusammenführung paralleler Zweige oder am Ende eines Subprozesses) werden die Variablen dieses Geltungsbereichs verworfen. Um Variablen in einen darüber liegenden Geltungsbereich zu übernehmen, müssen Sie ein Output-Mapping konfigurieren. Dabei wird die Aufrufhierarchie vom aktuellen Element ausgehend so lange nach oben durchlaufen, bis ein Geltungsbereich gefunden wird, in dem diese Variable bereits existiert. In diesen wird der neue Wert dann übernommen. Für den Fall, dass es sich um eine komplett neue Variable handelt, wird diese dann in den Geltungsbereich des Gesamtprozesses geschrieben.

> **Warnung**
> Bitte beachten Sie, dass ein Timer-Ereignis bei Benutzeraktivitäten einen eigenen Geltungsbereich erzeugt. Wenn Sie kein dediziertes Input-/Output-Mapping für die Benutzeraktivität definieren, werden bei Abschluss der Aktivität alle Variablen dieses Geltungsbereiches in den höheren Geltungsbereich übernommen.

Auswirkungen in der Prozessüberwachung

Bei der Ansicht eines Tokens werden immer die Variablen des aktuellen Geltungsbereichs angezeigt. Änderungen an den Variablen werden nur in diesen Geltungsbereich übernommen.

Die meisten Tokens besitzen einen eigenen Geltungsbereich. Die einzige Ausnahme bilden Benutzeraktivitäten, für die kein Input-/Output-Mapping konfiguriert wurde und die nicht parallel zu anderen Aktivitäten ausgeführt werden. Diese Tokens besitzen denselben Geltungsbereich wie das direkt darüber liegende Token.

In der Ansicht des Tokens ohne eigenen Geltungsbereich finden Sie eine Verknüpfung zum Token aus dem darüber liegenden Bereich. Wenn Sie diese Verknüpfung auswählen, können Sie die Variablen des verknüpften Geltungsbereichs anzeigen.

1.4.6. Verwenden von Timer-Ereignissen

Bei der Prozessmodellierung können Sie das BPMN-Element **intermediate boundary event** (angeheftetes Zwischenereignis) vom Typ Timer verwenden. Diese Zwischenereignisse können Sie an Benutzeraktivitäten oder Services heften. Zum Einstieg in das Arbeiten mit angehefteten Zwischenereignissen ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen.

Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="timer_event_process" name="Timer Event Process"
isExecutable="true">
    <startEvent id="start" />
        <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

Hinzufügen eines Zwischenereignisses

Fügen Sie dem Prozess das BPMN-Element **boundaryEvent** hinzu.

```
...
<process id="timer_event_process" name="Timer Event Process"
```

```
isExecutable="true">
  ...
  <userTask ... />
  <boundaryEvent id="timer" attachedToRef="task_hello_world">
    <timerEventDefinition>
      <timeDuration>PT1M</timeDuration>
    </timerEventDefinition>
  </boundaryEvent>
  ...
</process>
```

Erläuterung der Eigenschaften

- **id:**Die Eigenschaft wird als eindeutiger Bezeichner für die Sendenaktivität verwendet.
- **attachedToRef:** Die Eigenschaft definiert, an welches BPMN-Element dieses Zwischenereignis angeheftet werden soll.

Erläuterung der Ereignisdefinition:

- **timerEventDefinition:** Dieses BPMN-Element definiert, dass es sich bei dem Zwischenereignis um ein Zeitereignis handelt
- **timeDuration:** Dieses BPMN-Element definiert, dass es sich um eine relative Zeitspanne handelt. In diesem Fall ist eine Spanne von einer Minute definiert. Möchten Sie einen absoluten Zeitpunkt definieren, können Sie das BPMN-Element **timeDate** verwenden. In beiden Fällen müssen Sie die Werte im ISO-8601-Format angeben. Sie können die Werte auch aus Variablen ermitteln.

Hinzufügen eines Ereignispfades

Tritt im Prozess ein Zwischenereignis ein, folgt der Prozess einem separaten Prozesspfad. Sie können einen Prozess beispielsweise so konfigurieren, dass der Prozess von einem Zwischenereignis direkt zum Endereignis führt. Diese Konfiguration fügen Sie folgendermaßen in den Prozess ein:

```
...
<process id="timer_event_process" name="Timer Event Process"
isExecutable="true">
  ...
  <sequenceFlow id="s3" sourceRef="timer" targetRef="end" />
</process>
```

Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/
20100524/DC" xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
targetNamespace="" exporter="d.velop process modeler">
  <process id="timer_event_process" name="Timer Event Process"
isExecutable="true">
    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <boundaryEvent id="timer" attachedToRef="task_hello_world">
      <timerEventDefinition>
        <timeDuration>PT1M</timeDuration>
      </timerEventDefinition>
```

```
    </boundaryEvent>
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
    <sequenceFlow id="s3" sourceRef="timer" targetRef="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="timer_event_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="179" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="290" y="77" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="timer_di" bpmnElement="timer">
        <dc:Bounds x="372" y="139" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="462" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="215" y="117" />
        <di:waypoint x="290" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="390" y="117" />
        <di:waypoint x="462" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s3_di" bpmnElement="s3">
        <di:waypoint x="408" y="157" />
        <di:waypoint x="480" y="157" />
        <di:waypoint x="480" y="135" />
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

1.4.7. Verwenden von Verzweigungen

Beim Modellieren eigener Prozesse können Sie das BPMN-Element **Gateway** vom Typ exclusive (Verzweigung vom Typ exklusiv) verwenden, um dynamische Prozesse mit einer Entscheidungslogik zu erstellen.

Zum Einstieg in das Arbeiten mit exklusiven Verzweigungen ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Dieser Prozess sieht beispielsweise so aus:

Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:camunda="http://
camunda.org/schema/1.0/bpmn" targetNamespace="" exporter="d.velop process
modeler">
  <process id="exclusive_gateway_process" name="Exclusive Gateway Process"
isExecutable="true">
    <startEvent id="start" />
        <userTask id="task_hello_world" name="Hello World"
camunda:camundaUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

Diesen Beispielprozess erweitern Sie nun mit einer Verzweigung. Anhand des Wertes einer Prozessvariablen führt dieser Prozess entweder zu der Benutzeraktivität oder wird direkt beendet.

Hinzufügen einer Verzweigung

Fügen Sie dem Prozess zunächst das BPMN-Element **exclusiveGateway** hinzu. Das **exclusiveGateway**-Element repräsentiert eine Verzweigung mit nur einem einzigen möglichen Ausgangspfad. Ersetzen Sie die vorhandenen Sequenzflüsse anschließend mit den folgenden Sequenzflüssen:

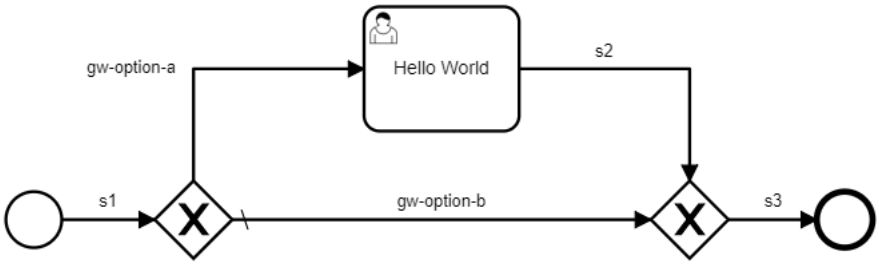
```
...
<process id="exclusive_gateway_process" name="Exclusive Gateway Process"
isExecutable="true">
  <startEvent id="start" />
  <exclusiveGateway id="gateway" default="gateway-option-b" />
  <userTask id="task_hello_world" name="Hello World" camunda:camundaUsers="$
{variables.get('dv_initiator')}" />
  <exclusiveGateway id="join" />
  <endEvent id="end" />
  <sequenceFlow id="s1" sourceRef="start" targetRef="gateway" />
  <sequenceFlow id="gateway-option-a" sourceRef="gateway"
targetRef="task_hello_world" />
  <sequenceFlow id="gateway-option-b" sourceRef="gateway" targetRef="join"
/>
  <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="join" />
  <sequenceFlow id="s3" sourceRef="join" targetRef="end" />
</process>
```

Erläuterung der Eigenschaften:

- **id:**Die Eigenschaft wird als eindeutiger Bezeichner für die Verzweigung verwendet.

- **default:** Die Eigenschaft definiert die ID des Sequenzflusses, der als Standard-Ausgangspfad der Verzweigung gewählt werden soll.

Der Prozess sieht dann beispielsweise so aus:



Hinzufügen einer Bedingung

Alle Ausgangspfade einer exklusiven Verzweigung müssen eine Bedingung enthalten. Mithilfe dieser Bedingung ermittelt die Anwendung, welchem Pfad der Prozess folgen soll. Für den Beispielprozess fügen Sie dem Sequenzfluss **gw-option-a** die Bedingung hinzu, dass dieser Pfad abhängig von der Variable **amount** gewählt wird. Entspricht diese Variable einem Wert größer 1000, soll der Sequenzfluss **gw-option-a** gewählt werden.

```
...
<sequenceFlow id="gw-option-a" sourceRef="gateway"
targetRef="task_hello_world" >
  <conditionExpression
xsi:type="tFormalExpression">#{variables.get("amount") > 1000}</
conditionExpression>
<sequenceFlow>
...
```

Um eine Prozessvariable verwenden zu können, müssen Sie diese der Prozessdefinition hinzufügen. In der Prozessdefinition fügen Sie die BPMN-Elemente **extensionElements** und **camunda:properties** zum BPMN-Element **process** hinzu. Zum Definieren der Prozessvariable fügen Sie dem BPMN-Element **camunda:properties** noch das Element **camunda:property** hinzu. In diesem Beispiel fügen Sie die Variable **amount** vom Typ **Number** ein. Diese ist eine Pflichtvariable, die beim Start mitgegeben werden muss.

```
...
<process id="exclusive_gateway_process" name="Exclusive Gateway Process"
isExecutable="true">
  <extensionElements>
    <camunda:properties>
      <camunda:property name="variable:amount*" value="Number!" />
    </camunda:properties>
  </extensionElements>
  ...
</process>
```

Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:camunda="http://
camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://www.omg.org/spec/BPMN/
20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
```

```
xmlns:di="http://www.omg.org/spec/DD/20100524/DI" targetNamespace=""
exporter="d.velop process modeler">
  <process id="exclusive_gateway_process" name="Exclusive Gateway Process"
isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="variable:amount*" value="Number!" />
      </camunda:properties>
    </extensionElements>
    <startEvent id="start" />
    <exclusiveGateway id="gateway" default="gateway-option-b" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <exclusiveGateway id="join" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="gateway" />
    <sequenceFlow id="gateway-option-a" sourceRef="gateway"
targetRef="task_hello_world">
      <conditionExpression
xsi:type="tFormalExpression">#{variables.get("amount") > 1000}</
conditionExpression>
    </sequenceFlow>
    <sequenceFlow id="gateway-option-b" sourceRef="gateway"
targetRef="join" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="join" />
    <sequenceFlow id="s3" sourceRef="join" targetRef="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1"
bpmnElement="exclusive_gateway_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="179" y="199" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="gateway_di" bpmnElement="gateway"
isMarkerVisible="true">
        <dc:Bounds x="275" y="192" width="50" height="50" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="410" y="80" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="join_di" bpmnElement="join"
isMarkerVisible="true">
        <dc:Bounds x="595" y="192" width="50" height="50" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="702" y="199" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="215" y="217" />
        <di:waypoint x="275" y="217" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="gateway-option-a_di" bpmnElement="gateway-option-
```

```
a">
        <di:waypoint x="300" y="192" />
        <di:waypoint x="300" y="120" />
        <di:waypoint x="410" y="120" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="gateway-option-b_di" bpmnElement="gateway-option-
b">
        <di:waypoint x="325" y="217" />
        <di:waypoint x="595" y="217" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="510" y="120" />
        <di:waypoint x="620" y="120" />
        <di:waypoint x="620" y="192" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s3_di" bpmnElement="s3">
        <di:waypoint x="645" y="217" />
        <di:waypoint x="702" y="217" />
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

Sie können den Wert der Variablen **amount** nun zum Beispiel beim Start des Prozesses belegen, und so den Prozessverlauf beeinflussen.

1.4.8. Einfügen von Benutzeraktivitäten

In diesem Thema erhalten Sie grundlegende Informationen über die Verwendung und Anpassung des BPMN-Elements **User Task** (Benutzeraktivität).

Wissenswertes zu Benutzeraktivitäten

Im BPMN-Standard gibt es unterschiedliche Konstrukte für die Zuweisung von Empfängern innerhalb eines **User Task** (Benutzeraktivität).

Wir empfehlen die Verwendung der Eigenschaft **camunda:candidateUsers**.

Potential Owner

```
<userTask id='theTask' name='important task' camunda:candidateUsers="$
{variables.get('myPerformer')}" />
```

Verwenden von Konstanten für Empfänger

Wenn Sie die Empfänger einer Aktivität als Konstante angeben möchten, verwenden Sie folgende Syntax:

```
${collections.from("identity:///identityprovider/scim/users/someUserId")}
// or
${collections.from("identity:///identityprovider/scim/users/someUserId",
"identity:///identityprovider/scim/users/someOtherUserId")}
```

Verwenden von Variablen für Empfänger

Wenn die Empfänger einer Aktivität aus Variablen ermittelt werden sollen, verwenden Sie folgende Syntax für die Referenzierung der Variablen:

```
${variables.get("variableName")}
```

Verlinken von Zusatzinformationen zu Benutzeraktivitäten

Um Benutzeraktivitäten Zusatzinformationen für die Bearbeitung hinzuzufügen, können Sie die dafür reservierte Prozessvariable **dv_attachment** verwenden.

In dieser Variable können Sie eine URL speichern. Die URL wird bei Erstellung einer Aktivität hinzugefügt, wenn Sie sie zuvor eingetragen haben.

> **Warnung**
> Die Prozessvariable **dv_attachment** ist mit dem Datentyp **DmsObject** vordefiniert. Möchten Sie eine andere URI als die zu einem Objekt der DMS-App verwenden, müssen Sie den Wert des Datentyps auf **String** ändern.

Hinzufügen von Metadaten zu Benutzeraktivitäten

Um Metadaten zu Benutzeraktivitäten hinzuzufügen, können Sie die Metadaten innerhalb des BPMN-Elements **userTask** definieren. Fügen Sie das Element **extensionElements** und darunter das Element **camunda:properties** hinzu, falls die Prozessdefinition diese Elemente noch nicht enthält. Unterhalb des Elements **camunda:properties** können Sie nun wie folgt Metadaten definieren:

```
<userTask>
...
        <extensionElements>
                <camunda:properties>
                        <camunda:property
name="metadata:invoiceNumber:value" value="$
{variables.get("invoiceNumber")}" />
                <camunda:property name="metadata:invoiceNumber:caption"
value="Invoice Number" />
                <camunda:property name="metadata:invoiceNumber:caption:de"
value="Rechnungsnummer" />
                <camunda:property name="metadata:invoiceNumber:type"
value="String" />
        </camunda:properties>
        </extensionElements>
...
</userTask>
```

Das Attribut **name** innerhalb des Elements **camunda:property** kann folgende Werte annehmen:

Wert	Bedeutung	
metadata:<key>:value	Definiert den Wert eines Metadatums.	Der Key des Metadatums muss folgendem Ausdruck entsprechen: [A-Za-z0-9]{1,255}
metadata:<key>:caption	(Optional) Definiert die Standardbeschriftung eines Metadatums. Falls nicht vorhanden, wird der Schlüssel als Beschriftung verwendet.	
metadata:<key>:caption:<language>	(Optional) Definiert die Beschriftung eines Metadatums für die angegebene Sprache. Das Sprachkürzel muss der ISO-639-1 Norm entsprechend aus zwei Buchstaben bestehen. Falls nicht vorhanden, wird die Standardbeschriftung bzw. der Schlüssel verwendet.	
metadata:<key>:type	(Optional) Definiert den Datentyp des Metadatums.	Entspricht dem Datentyp aus der TaskApp (**String**, **Number**, **Money**, **Date**). Wenn kein Typ angegeben wird, wird der Typ **String** angenommen.

Das Attribut **value** kann entweder einen festen Wert oder einen Ausdruck enthalten. Dieser Wert darf maximal 255 Zeichen lang sein. Bei der Verwendung eines Ausdrucks gilt diese Beschränkung für das

Ergebnis des Ausdrucks. Darüber hinaus muss das Ergebnis eines Ausdrucks ein einzelner Wert sein. Werte vom Typ **Collection** oder **Array** sind nicht erlaubt.

Bestimmen der Aufbewahrungsdauer einer Benutzeraktivität

Sie können die Aufbewahrungsdauer von Benutzeraktivitäten innerhalb des BPMN-Elements **userTask** definieren. Fügen Sie das Element **extensionElements** und darunter das Element **camunda:properties** hinzu, falls die Prozessdefinition diese Elemente noch nicht enthält. Unterhalb des Elements **camunda:properties** können Sie nun wie folgt die Aufbewahrungsdauer definieren:

```
<userTask>
...
        <extensionElements>
                <camunda:properties>
                        <camunda:property name="retentionTime" value="P7D"
/>
                </camunda:properties>
        </extensionElements>
...
</userTask>
```

Das Attribut **value** kann entweder einen festen Wert oder einen Ausdruck enthalten. Definieren Sie die Aufbewahrungsdauer als Zeitspanne nach ISO-8601 in Tagen, z.B. **P30D** für 30 Tage.

Bestimmen der Nutzbarkeit von Aktionen in der Benutzeroberfläche einer Benutzeraktivität

Sie können die Nutbarkeit von Aktionen in der Benutzeroberfläche der Benutzeraktivität innerhalb des BPMN-Elements **userTask** definieren. Fügen Sie das Element **extensionElements** und darunter das Element **camunda:properties** hinzu, falls die Prozessdefinition diese Elemente noch nicht enthält. Unterhalb des Elements **camunda:properties** können Sie nun wie folgt die Aufbewahrungsdauer definieren:

```
<userTask>
...
        <extensionElements>
                <camunda:properties>
                        <camunda:property name="actionScope:complete"
value="[list, details]" />
                </camunda:properties>
        </extensionElements>
...
</userTask>
```

Die Eigenschaft **name** des Elements enthält das Prefix **actionScope** gefolgt von der zu konfigurierenden Aktion (in diesem Beispiel **complete**). Die Eigenschaft **value** enthält die gewünschten Optionen für die Nutzbarkeit. Welche Aktionen und Nutzbarkeitsoptionen möglich sind, können Sie der API-Dokumentation der TaskApp entnehmen.

Hinzufügen von Benutzeroberflächen zu einer Benutzeraktivität

Wenn Sie bei der Erstellung eines Prozessmodells das BPMN-Element **User Task** (Benutzeraktivität) verwenden, können Sie der Aktivität eine Benutzeroberfläche hinzufügen. Dem Benutzer wird dann beispielsweise eine Schaltfläche angezeigt, mit der er eine Aufgabe beginnen kann.

Um eine Benutzeroberfläche hinzuzufügen, verwenden Sie die Eigenschaft **f ormKey**.

Beispiel

Form Key

```
<userTask id="someTask" camunda:formKey="uri:/myapp/myform">
  ...
</userTask>
```

Der Schlüssel besteht aus einem Präfix und einem Wert. Diese werden durch einen Doppelpunkt voneinander getrennt. Folgende Präfixe können verwendet werden:

Präfix	Bedeutung	Beispiel
uri	Die Benutzeroberfläche wird über eine URI angegeben.	uri:https://example.com/

Verwenden von Prozessvariablen

Der Schlüssel kann auch Prozessvariablen beinhalten. Diese können in Form von Expression Language angegeben werden.

Beispiel

```
uri:${variables.get("formUri")}

uri:https://example.com/?queryParam=${variables.get("formParam")}
```

Externer Zugriff auf Prozessvariablen

Mit dem Ausdruck **${process.task.variablesUri}** kann ein URI erzeugt werden, um einer externen Applikation den Zugriff auf die aktuellen Variablen dieser Benutzeroberfläche zu ermöglichen. Mittels der HTTP-Methoden **GET** und **PUT** können diese Variablen gelesen und verändert werden. In dieser Bearbeitungsoberfläche muss zu den verwendenden Variablen ein Input- und Output-Mapping definiert sein.

Beispiel

```
uri:/myApp?variablesUri=${process.task.variablesUri}
```

Verwenden von Prozessvariablen in einer Benutzeroberfläche

Enthält eine Benutzeraktivität ein Formular, können Sie mit HTTP lesend und schreibend auf die verwendeten Prozessvariablen zugreifen.

Zum Einstieg in das Arbeiten mit Prozessvariablen in einem Formular ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Dieser Prozess sieht beispielsweise so aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="GettingStarted" name="Example: Getting started"
isExecutable="true">
    <startEvent id="start"/>
    <userTask id='task_hello_world' name='Hello World'
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end"/>
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

Zur Vereinfachung sind die grafischen Informationen zum BPMN-Diagramm in dieser BPMN-Beispieldatei nicht enthalten.

Erstellen eines Formulars

Erstellen Sie zunächst eine HTML-Datei. Diese kann wie folgt aussehen. Die Code-Beispiele sind auf die Funktionsweise in Google Chrome ausgerichtet. Andere Browser können möglicherweise eine andere Syntax erfordern.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Getting started</title>
</head>
<body/>
</html>
```

Legen Sie die HTML-Datei so ab, dass sie mit einem Http-Gateway erreichbar ist. Fügen Sie den URI des HTML-Dokuments zum BPMN-Element **userTask** hinzu.

```
...
<process id="GettingStarted" name="Example: Getting started"
isExecutable="true">
        ...
    <userTask id='task_hello_world'
name='Hello World' camunda:formKey="uri:/res/process/process-
form.html?variables=${process.task.variablesUri}" camunda:candidateUsers="$
{variables.get('dv_initiator')}" />
    ...
```

Erläuterung der Eigenschaft

- **camunda:formKey**: In dieser Eigenschaft wird der URI definiert, der die Oberfläche für die Benutzeraktivität bereitstellt. Das Präfix **uri:** ist dabei verpflichtend. Die Variable **${process.task.variablesUri}** wird als Query-Parameter zum URI ergänzt. Die Variable wird von der Anwendung zur Laufzeit in einen URI aufgelöst, der einen HTTP-Endpunkt zum Lesen und Schreiben der Variablen bereitstellt.

Erstellen der Benutzeroberfläche

Erstellen Sie zunächst das Grundgerüst für eine HTML-Tabelle in Ihrem HTML-Dokument. Im späteren Verlauf werden die Variablen dann in dieser Tabelle angezeigt.

```
...
<body>
    <table id="variables">
                <thead>
                        <tr>
                                <th>Name</th>
                                <th>Value</th>
                        </tr>
                </thead>
                <tbody id="variableValues">
                </tbody>
        </table>
</body>
...
```

Abfragen der Prozessvariablen

Im Head des HTML-Dokuments erstellen Sie eine Funktion zum Auslesen der Prozessvariablen. Die Prozessvariablen werden mit einer HTTP-GET-Anforderung entsprechend der Rest-API von der Anwendung abgefragt. Das Laden der Variablen wird durch das **onLoad**-Event des **body**-Elements ausgelöst.

```
...
<head>
        ...
        <script type="text/javascript">

                var urlParams = new URLSearchParams(window.location.search);
                var variablesUri = urlParams.get('variables');

                function loadVariables() {
                        fetch(variablesUri)
                                .then(response => response.json());
                }

        </script>
</head>
<body onload="loadVariables()">
...
```

Nachdem die Variablen geladen wurden, können Sie diese nun in der vorbereiteten Tabelle anzeigen. Im folgenden Beispiel wird die JavaScript-Bibliothek jQuery verwendet. Fügen Sie im Head des HTML-Dokuments die jQuery-Bibliothek hinzu und implementieren Sie entsprechende Methoden zum Anzeigen der Variablen.

```
...
<head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/
jquery.min.js"></script>
        <script type="text/javascript">

                var urlParams = new URLSearchParams(window.location.search);
                var variablesUri = urlParams.get('variables');

                function loadVariables() {
                        fetch(variablesUri)
                                .then(response => response.json())
                                .then(json =>
createTableFromVariables(json.variables || {}));
                }

                function createTableFromVariables(variables) {
                        Object.keys(variables).forEach(variable => {
                                var value = variables[variable];
                                $('#variableValues').append(
`<tr class="variable">${createKeyColumn(variable)}$
{createValueColumn(value)}</tr>`);
                        });
                }

                function createKeyColumn(variable) {
                        return `<td><span class="key">${variable}</span></
td>`;
                }

                function createValueColumn(value) {
                        return `<td><input class="value" type="text"
value="${value}"></td>`;
```

```
                }

        </script>
</head>
<body onload="loadVariables()">
...
```

Die Variablen werden nun nach dem Laden des HTML-Dokuments abgefragt und angezeigt.

Schreiben der Prozessvariablen

Fügen Sie in den Skript-Bereich Methoden ein, die das Zurückschreiben der Variablen in die Anwendung ermöglichen. Anschließend fügen Sie einen **button** unterhalb der Tabelle hinzu, der die Methode **saveVariables** aufruft.

```
...
<head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/
jquery.min.js"></script>
        <script type="text/javascript">

                var urlParams = new URLSearchParams(window.location.search);
                var variablesUri = urlParams.get('variables');

                ...

                function saveVariables() {
                        var variables = createVariablesFromTable();

                        // These variables are read-only and must not be
sent to server when setting variables
                        delete variables['dv_initiator'];
                        delete variables['dv_start_time'];

                    const headers = new Headers();
                        headers.append('Cache-Control', 'no-cache');
                        headers.append('Content-Type', 'application/json');

                        let promise = fetch(variablesUri, {
                                method: 'PUT',
                                headers: headers,
                                credentials: 'same-origin',
                                body: JSON.stringify({variables})
                        });

                        promise.then(function(response) {
                                let status = response.status;
                                if (status !== 200 ) {
                                        window.alert("Saving variables
failed: " + status);
                                } else {
                                        window.alert("Variables saved");
                                }

                        });
                }
```

```
                function createVariablesFromTable() {
                        var variables = {};
                        $("tr.variable").each(function() {
                                $this = $(this);
                                var key = $this.find("span.key").html();
                                var value = $this.find("input.value").val();
                                variables[key] = value;
                        });
                        return variables;
                }

        </script>
</head>
<body onload="loadVariables()">
        ...
        <br/>
        <button onClick="saveVariables()">Save</button>
</body>
```

Mithilfe des Formulars, das bei der Ausführung des Prozesses angezeigt wird, können Sie die aktuellen Prozessvariablen anzeigen und ändern.

Das finale HTML-Dokument sieht wie folgt aus. Das HTML-Dokument enthält einige ergänzende CSS-Definitionen zur optimierten Darstellung der Oberflächenelemente.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Process Form</title>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/
jquery.min.js"></script>
        <style>
                table {
                  border-collapse: collapse;
                  width: 100%;
                }

                table, th, td {
                  border: 1px solid black;
                }

                td {
                  padding: 3px;
                }

                .key {
                        color: gray;
                }

                th {
                  text-align: left;
                }

                input {
```

```
                        width: 100%;
                        border: 0;
                }
        </style>
    <script type="text/javascript">

                var urlParams = new URLSearchParams(window.location.search);
                var variablesUri = urlParams.get('variables');

                function loadVariables() {
                        fetch(variablesUri)
                                .then(response => response.json())
                                .then(json =>
createTableFromVariables(json.variables || {}));
                }

                function saveVariables() {
                        var variables = createVariablesFromTable();

                        // These variables are read-only and must not be
sent to server when setting variables
                        delete variables['dv_initiator'];
                        delete variables['dv_start_time'];

                const headers = new Headers();
                        headers.append('Cache-Control', 'no-cache');
                        headers.append('Content-Type', 'application/json');

                        let promise = fetch(variablesUri, {
                                method: 'PUT',
                                headers: headers,
                                credentials: 'same-origin',
                                body: JSON.stringify({variables})
                        });

                        promise.then(function(response) {
                                let status = response.status;
                                if (status !== 200 ) {
                                        window.alert("Saving variables
failed: " + status);
                                } else {
                                        window.alert("Variables saved");
                                }

                        });
                }

                function createTableFromVariables(variables) {
                        Object.keys(variables).forEach(variable => {
                                var value = variables[variable];
                                $('#variableValues').append(
`<tr class="variable">${createKeyColumn(variable)}$
{createValueColumn(value)}</tr>`);
                        });
                }
```

```
                function createKeyColumn(variable) {
                        return `<td><span class="key">${variable}</span></
td>`;
                }

                function createValueColumn(value) {
                        return `<td><input class="value" type="text"
value="${value}"></td>`;
                }

                function createVariablesFromTable() {
                        var variables = {};
                        $("tr.variable").each(function() {
                                $this = $(this);
                                var key = $this.find("span.key").html();
                                var value = $this.find("input.value").val();
                                variables[key] = value;
                        });
                        return variables;
                }

        </script>
</head>
<body onload="loadVariables()">
    <table id="variables">
                <thead>
                        <tr>
                                <th>Name</th>
                                <th>Value</th>
                        </tr>
                </thead>
                <tbody id="variableValues">
                </tbody>
        </table>
        <br/>
        <button onClick="saveVariables()">Save</button>
</body>
</html>
```

1.4.9. Verwenden einer Multi-Instanz

Sie können Mehrfachausführungen zu Benutzeraktivitäten und Subprozessen hinzufügen. Voraussetzung dafür ist, dass die Aktivität oder der Prozess auf einer Variable mit Mehrfachwert basiert. Für jeden Wert der Variable wird eine Ausführung der Aktivität vorgenommen.

Zum Einstieg in das Arbeiten mit Multi-Instanzen ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Dieser Prozess sieht beispielsweise so aus:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="multi_instance_process" name="Multi Instance Process"
isExecutable="true">
    <startEvent id="start" />
        <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

Hinzufügen einer Multi-Instanz

Fügen Sie der Benutzeraktivität das BPMN-Element **multiInstanceLoopCharacteristics** hinzu. Ändern Sie anschließend die Variable, die den Empfänger der Benutzeraktivität bestimmt, in **myPerformer**.

```
...
<process id="multi_instance_process" name="Multi Instance Process"
isExecutable="true">
  ...
  <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('myPerformer')}" >
      <multiInstanceLoopCharacteristics
          isSequential="false"
          camunda:collection="${variables.get('performers')}"
          camunda:elementVariable="myPerformer" />
  </userTask>
  ...
</process>
```

Erläuterung der Eigenschaften

- **isSequential**: Verwenden Sie den Wert **true**, um die Aktivitäten nacheinander auszuführen und **false**, um sie gleichzeitig auszuführen.
- **camunda:collection**: Ausdruck, der eine Mehrfachvariable zurückgibt.
- **camunda:elementVariable**: Variablenname, der lokal für den jeweiligen Wert verwendet wird. Für die Variable muss eine Definition vorhanden sein.

Im Beispielprozess wird für jeden Wert der Variable **performers** eine Aktivität zugestellt. Mit der lokalen Variable **myPerformer** kann jedes einzelne Element als Empfänger angegeben werden.

Damit die Variablen **performers** und **myPerformer** verwendet werden können, müssen Sie diese definieren. Fügen Sie die BPMN-Elemente **extensionElements** und **camunda:properties** zum BPMN-Element **process** hinzu. Zum Definieren der Prozessvariablen fügen Sie den BPMN-Elementen **camunda:properties** noch jeweils das Element **camunda:property** hinzu.

```
...
<process id="multi_instance_process" name="Multi Instance Process"
isExecutable="true">
  <extensionElements>
    <camunda:properties>
      <camunda:property name="variable:performers*" value="[Identity]!" />
      <camunda:property name="variable:myPerformer" value="Identity"
/>
```

```
    </camunda:properties>
  </extensionElements>
  ...
</process>
```

Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/
20100524/DC" xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
targetNamespace="" exporter="d.velop process modeler">
  <process id="multi_instance_process" name="Multi Instance Process"
isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="variable:performers*" value="[Identity]!" />
        <camunda:property name="variable:myPerformer" value="Identity"
/>
      </camunda:properties>
    </extensionElements>
    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('myPerformer')}" >
      <multiInstanceLoopCharacteristics
            isSequential="false"
            camunda:collection="${variables.get('performers')}"
            camunda:elementVariable="myPerformer" />
    </userTask>
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="multi_instance_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="179" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="290" y="77" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="462" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="215" y="117" />
        <di:waypoint x="290" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="390" y="117" />
        <di:waypoint x="462" y="117" />
```

```
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

Den Wert der Variablen **performers** müssen Sie nun beim Start des Prozesses belegen.

1.4.10. Verwenden eines Startformulars

Sie können ein Formular zu einem Startereignis hinzufügen, wie beispielsweise einen Urlaubsantrag. Sie können das Startformular dann als Einstiegspunkt für diesen Prozess verwenden. Um einem Startereignis ein Formular hinzuzufügen, verwenden Sie die Eigenschaft **formKey**.

```
<startEvent id="start" name="start" camunda:formKey="uri:/myapp/mystart">
    ...
</startEvent>
```

1.4.11. Verwenden von Services

In diesem Thema erfahren Sie, wie Sie Services in Ihre Prozesse einbinden können. Beim Einfügen von Services können Sie sich entscheiden, ob der Service synchron oder asynchron durchgeführt wird.

Verwenden eines synchronen Service

Automatisierte Aktivitäten werden in einem Prozess mithilfe von Services ausgeführt. Für die Verwendung von synchronen Services in BPMN können Sie das BPMN-Element **Send Task** (Sendenaktiviät) verwenden. Erreicht der Prozess die Sendenaktivität, so werden die im Prozess definierten Daten an den HTTP-Endpunkt des Serviceanbieters geschickt und unmittelbar auf das Ergebnis gewartet. Anschließend wird die Prozessausführung direkt fortgesetzt. Dieser Artikel zeigt Ihnen anhand eines einfachen Beispielservice, wie Sie einen synchronen Service verwenden können. Der Beispielservice "Hello World" wandelt einen beliebigen Text in "Hello <yourname>" um, wobei <yourname> der Text ist, der zuvor an den Service gesendet wurde.

Um dieses Beispiel durchzuführen, ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

Hinzufügen einer Sendenaktivität

Fügen Sie zunächst eine Sendenaktivität nach dem Startereignis hinzu.

```
...
<process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
  <startEvent id="start" />
  <sendTask id="call_service"
```

```
        name="Call Service 'Hello World'"
        camunda:asyncBefore="true"
            camunda:asyncAfter="true"
        camunda:delegateExpression="${syncService}"
        camunda:exclusive="true">
  </sendTask>
  ...
</process>
```

Erläuterung der Eigenschaften:

- **id**: Die Eigenschaft wird als eindeutiger Bezeichner für die Sendenaktivität verwendet.
- **name**: Die Eigenschaft wird als Anzeigename in den Benutzeroberflächen verwendet.
- **camunda:***: Diese Eigenschaften beinhalten technische Informationen, die zur Ausführung benötigt werden.

> **Anmerkung**
> Bei **synchronen Services** müssen Sie für die **camunda**-Eigenschaften immer die Werte des Beispiels eintragen. Wenn Sie abweichende Werte eintragen, kann der Prozess nicht bereitgestellt werden.

Fügen Sie nun der Sendenaktivität die BPMN-Elemente **extensionElements** und **camunda:inputOutput** hinzu. Sie müssen alle Daten definieren, die an den HTTP-Endpunkt des Services gesendet werden sollen. Zur Definition der Daten fügen Sie für jeden Wert das BPMN-Element **camunda:inputParameter** in die Prozessdefinition ein. Anschließend müssen Sie auch die erwarteten Ausgabewerte definieren. Fügen Sie hierzu für jeden Wert das BPMN-Element **camunda:outputParameter** in die Prozessdefinition ein.

```
...
<process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
  <startEvent id="start" />
  <sendTask id="call_service" ...>
    <extensionElements>
      <camunda:inputOutput>
        <camunda:inputParameter name="service.uri">/process/services/
helloworld/sync</camunda:inputParameter>
        <camunda:inputParameter name="yourName">$
{variables.getDisplayValue("dv_initiator")}</camunda:inputParameter>
        <camunda:outputParameter name="greeting">$
{variables.get("greeting")}</camunda:outputParameter>
      </camunda:inputOutput>
    </extensionElements>
  </sendTask>
  ...
</process>
```

Erläuterung der Parameter:

- **service.uri**: Dieser Parameter muss immer eingefügt werden. Er definiert den URI, unter dem der HTTP-Endpunkt des Services erreichbar ist. Das Beispiel zeigt den URI des "Hello World"-Services. Der Wert dieser Variablen muss eine Konstante sein, das heißt er darf keine Ausdrücke enthalten.
- **yourName**: Dieser Parameter wurde vom konkreten Service als Eingabewert definiert. In diesem Beispiel wird der Anzeigename des Benutzers verwendet, der den Prozess gestartet hat.
- **greeting**: Der Service hat als einzigen Ausgabewert den Parameter **greeting** definiert. Dieser Parameter wird bei der Antwort des Services automatisch in eine gleichnamige Variable im Geltungsbereich

der Sendenaktivität geschrieben. Der Parameter definiert, dass der Inhalt dieses Ergebniswertes bei Fortführung des Prozesses in die Variable **greeting** im Geltungsbereich des gesamten Prozesses geschrieben wird.

Um die Kommunikation zwischen der Anwendung und dem Service zu ermöglichen, müssen Sie die Schnittstelle des Services zum Prozess hinzufügen. Zum Hinzufügen der Schnittstelle tragen Sie die BPMN-Elemente **extensionElements** und **camunda:properties** in die Prozessdefinition ein. Anschließend fügen Sie je ein BPMN-Element **camunda:property** für den Service-Eingabewert **yourName**, den Service-Ausgabewert **greeting**, sowie die resultierende Prozessvariable **greeting** ein. Alle Werte sind vom Datentyp **String**.

```
...
<process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
  <extensionElements>
    <camunda:properties>
      <camunda:property name="service:/process/services/helloworld/
sync:in:yourName" value="String" />
      <camunda:property name="service:/process/services/helloworld/
sync:out:greeting" value="String" />
          <camunda:property name="service:/process/services/helloworld/
sync:name" value="Hello World" />
      <camunda:property name="variable:greeting" value="String" />
    </camunda:properties>
  </extensionElements>
  ...
</process>
```

Die Variable **greeting** enthält nun den Text "Hello <display name of process start user>". Verwenden Sie diese Variable nun als Namen für die Benutzeraktivität, damit der Anwender den freundlichen Gruß als Betreff der zu erledigenden Aufgabe erhält.

```
...
<process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
  ...
  <userTask id="task_hello_world" name="${variables.get('greeting')}"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
  ...
</process>
```

Damit die Prozessaktivitäten in der richtigen Reihenfolge ausgeführt werden, müssen Sie die Sequenzflusselemente noch an die neuen Aktivitäten anpassen.

```
<process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
  ...
  <sequenceFlow id="s1" sourceRef="start" targetRef="call_service" />
  <sequenceFlow id="s2" sourceRef="call_service"
targetRef="task_hello_world" />
  <sequenceFlow id="s3" sourceRef="task_hello_world" targetRef="end" />
</process>
```

Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:di="http://www.omg.org/spec/DD/
20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
targetNamespace="" exporter="d.velop process modeler">
  <process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="service:/process/services/helloworld/
sync:in:yourName" value="String" />
        <camunda:property name="service:/process/services/helloworld/
sync:out:greeting" value="String" />
        <camunda:property name="service:/process/services/helloworld/
sync:name" value="Hello World" />
        <camunda:property name="variable:greeting" value="String" />
      </camunda:properties>
    </extensionElements>
    <startEvent id="start" />
    <sendTask id="call_service"
          name="Call Service 'Hello World'"
          camunda:asyncBefore="true"
          camunda:asyncAfter="true"
          camunda:delegateExpression="${syncService}"
          camunda:exclusive="true">
      <extensionElements>
        <camunda:inputOutput>
          <camunda:inputParameter name="service.uri">/process/services/
helloworld/sync</camunda:inputParameter>
          <camunda:inputParameter name="yourName">$
{variables.getDisplayValue("dv_initiator")}</camunda:inputParameter>
          <camunda:outputParameter name="greeting">$
{variables.get("greeting")}</camunda:outputParameter>
        </camunda:inputOutput>
      </extensionElements>
    </sendTask>
    <userTask id="task_hello_world" name="${variables.get('greeting')}"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="call_service" />
    <sequenceFlow id="s2" sourceRef="call_service"
targetRef="task_hello_world" />
    <sequenceFlow id="s3" sourceRef="task_hello_world" targetRef="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane" bpmnElement="sync_service_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="173" y="102" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="209" y="120" />
        <di:waypoint x="250" y="120" />
      </bpmndi:BPMNEdge>
```

```
      <bpmndi:BPMNShape id="call_service_di" bpmnElement="call_service">
        <dc:Bounds x="250" y="80" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="350" y="120" />
        <di:waypoint x="390" y="120" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="390" y="80" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s3_di" bpmnElement="s3">
        <di:waypoint x="490" y="120" />
        <di:waypoint x="532" y="120" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="532" y="102" width="36" height="36" />
      </bpmndi:BPMNShape>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

Verwenden eines asynchronen Service

Automatisierte Aktivitäten werden in einem Prozess mithilfe von Services ausgeführt. Für die Verwendung von asynchronen Services in BPMN können Sie die BPMN-Elemente **Send Task** (Sendenaktivität) und **Receive Task** (Empfangenaktiviät) verwenden. Erreicht der Prozess die Sendenaktivität, so werden die im Prozess definierten Daten an den HTTP-Endpunkt des Serviceanbieters geschickt. Anschließend wird die Prozessausführung so lange pausiert, bis der Serviceanbieter das Ergebnis der Verarbeitung zurückmeldet. Dieser Artikel zeigt Ihnen anhand eines einfachen Beispielservice, wie Sie einen asynchronen Service verwenden können. Der Beispielservice "Hello World" wandelt einen beliebigen Text in "Hello <yourname>" um, wobei <yourname> der Text ist, der zuvor an den Service gesendet wurde.

Um dieses Beispiel durchzuführen, ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="async_service_process" name="Async Service Process"
isExecutable="true">
    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

Hinzufügen einer Sendenaktivität

Fügen Sie zunächst eine Sendenaktivität nach dem Startereignis hinzu.

```
...
<process id="async_service_process" name="Async Service Process"
```

```
isExecutable="true">
  <startEvent id="start"/>
  <sendTask id="call_service"
        name="Call Service 'Hello World'"
        camunda:asyncBefore="true"
        camunda:delegateExpression="${asyncService}"
        camunda:exclusive="true">
  </sendTask>
  ...
</process>
```

Erläuterung der Eigenschaften:

- **id**: Die Eigenschaft wird als eindeutiger Bezeichner für die Sendenaktivität verwendet.
- **name**: Die Eigenschaft wird als Anzeigename in den Benutzeroberflächen verwendet.
- **camunda:***: Diese Eigenschaften beinhalten technische Informationen, die zur Ausführung benötigt werden.

> **Anmerkung**
> Bei **asynchronen Services** müssen Sie für die **camunda**-Eigenschaften immer die Werte des Beispiels eintragen. Wenn Sie abweichende Werte eintragen, kann der Prozess nicht bereitgestellt werden.

Fügen Sie nun der Sendenaktivität die BPMN-Elemente **extensionElements** und **camunda:inputOutput** hinzu. Sie müssen alle Daten definieren, die an den HTTP-Endpunkt des Services gesendet werden sollen. Zur Definition der Daten fügen Sie für jeden Wert das BPMN-Element **camunda:inputParameter** in die Prozessdefinition ein.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  <startEvent id="start"/>
  <sendTask id="call_service" ...>
    <extensionElements>
      <camunda:inputOutput>
        <camunda:inputParameter name="service.uri">/process/services/
helloworld/async</camunda:inputParameter>
        <camunda:inputParameter name="yourName">$
{variables.getDisplayValue("dv_initiator")}</camunda:inputParameter>
                <camunda:inputParameter name="service.callbackBase">/</
camunda:inputParameter>
          </camunda:inputOutput>
    </extensionElements>
  </sendTask>
  ...
</process>
```

Erläuterung der Parameter:

- **service.uri**: Dieser Parameter ist obligatorisch und muss daher immer eingefügt werden. Der Parameter definiert den URI, unter dem der HTTP-Endpunkt des Services erreichbar ist. Das Beispiel zeigt den URI des "Hello World"-Services. Der Wert dieser Variablen muss eine Konstante sein, das heißt, der Wert darf keine Ausdrücke enthalten.
- **yourName**: Dieser Parameter wird vom konkreten Service als Eingabewert definiert. In diesem Beispiel wird der Anzeigename des Benutzers verwendet, der den Prozess gestartet hat.

Änderung der Antwort-URIs

Sie können die Origin der URIs ändern, die dem Service für die Antwort übergeben werden. Hierfür stehen Ihnen die folgenden zwei Optionen zur Verfügung.

Angeben der Origin am SendTask eines konkreten Services

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  <startEvent id="start"/>
  <sendTask id="call_service" ...>
    <extensionElements>
      <camunda:inputOutput>
        ...
                <camunda:inputParameter name="service.callbackBase">/</
camunda:inputParameter>
                ...
          </camunda:inputOutput>
    </extensionElements>
  </sendTask>
  ...
</process>
```

Fügen Sie dem SendTask einen weiteren **camunda:inputParameter** mit dem Namen **service.callbackBase** hinzu. Dieser Parameter gilt dann für diesen konkreten Serviceaufruf.

Angeben der Origin für den gesamten Prozess

Tragen Sie die BPMN-Elemente **extensionElements** und **camunda:properties** in die Prozessdefinition ein.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  <extensionElements>
    <camunda:properties>
          ...
      <camunda:property name="service:/process/services/helloworld/
async:callbackBase" value="/" />
      <!-- alternative <camunda:property name="service:*:callbackBase"
value="/" /> -->
      ...
    </camunda:properties>
  </extensionElements>
  ...
</process>
```

Erstellen Sie ein Element **camunda:property** mit dem Namen **service:<service>:callbackBase**. Der Platzhalter **<service>** entspricht der URI des Services, oder * für alle Services.

Die angegebene Origin muss in beiden Fällen dem Format **https://domain[:port]** entsprechen. Alternativ entspricht die Angabe von / einer relativen URI.

Hinzufügen einer Empfangenaktivität

Fügen Sie der Prozessdefinition nun die Empfangenaktivität hinzu. Mithilfe der Empfangenaktivität können die verarbeiteten Daten vom Service an den Prozess übermittelt werden.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  ...
  </sendTask>
  <receiveTask id="receive_service_response"
        name="Wait for Service 'Hello World'"
        camunda:asyncAfter="true"
        camunda:exclusive="true">
  </receiveTask>
  ...
</process>
```

Erläuterung der Eigenschaften:

- **id**: Die Eigenschaft wird als eindeutiger Bezeichner für die Sendenaktivität verwendet.
- **name**: Die Eigenschaft wird als Anzeigename in den Benutzeroberflächen verwendet.
- **camunda:***: Diese Eigenschaften beinhalten technische Informationen, die zur Ausführung benötigt werden. Bei asynchronen Services müssen Sie für die Eigenschaft **camunda:asyncAfter** immer den Wert **true** eintragen.

Fügen Sie der Empfangenaktivität die BPMN-Elemente **extensionElements** und **camunda:inputOutput** hinzu. Sie müssen alle Daten definieren, die vom HTTP-Endpunkt des Services als Antwort erwartet werden. Zur Definition der Daten fügen Sie für jeden Wert das BPMN-Element **camunda:outputParameter** ein.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  ...
  </sendTask>
  <receiveTask id="receive_service_response" ...>
    <extensionElements>
      <camunda:inputOutput>
        <camunda:outputParameter name="greeting">$
{variables.get("greeting")}</camunda:outputParameter>
      </camunda:inputOutput>
    </extensionElements>
  </receiveTask>
  ...
</process>
```

Erläuterung der Parameter:

- **greeting**: Der Service hat als einzigen Ausgabewert den Parameter **greeting** definiert. Dieser Parameter wird bei der Antwort des Services automatisch in eine gleichnamige Variable im Geltungsbereich der Empfangenaktivität geschrieben. Der Parameter definiert, dass der Inhalt dieses Ergebniswertes bei Fortführung des Prozesses in die Variable **greeting** im Geltungsbereich des gesamten Prozesses geschrieben wird.

Um die Kommunikation zwischen der Anwendung und dem Service zu ermöglichen, müssen Sie die Schnittstelle des Services zum Prozess hinzufügen. Zum Hinzufügen der Schnittstelle tragen Sie, sofern nicht bereits in vorherigen Schritten geschehen, die BPMN-Elemente **extensionElements** und **camunda:properties** in die Prozessdefinition ein. Anschließend fügen Sie je ein BPMN-Element **camunda:property** für den Service-Eingabewert **yourName**, den Service-Ausgabewert **greeting**, sowie die resultierende Prozessvariable **greeting** ein. Alle Werte sind vom Datentyp **String**.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  <extensionElements>
    <camunda:properties>
      <camunda:property name="service:/process/services/helloworld/
async:in:yourName" value="String" />
      <camunda:property name="service:/process/services/helloworld/
async:out:greeting" value="String" />
      <camunda:property name="service:/process/services/helloworld/
async:name" value="Hello World" />
      <camunda:property name="variable:greeting" value="String" />
    </camunda:properties>
  </extensionElements>
  ...
</process>
```

Die Variable **greeting** enthält nun den Text "Hello <display name of process start user>". Verwenden Sie diese Variable nun als Namen für die Benutzeraktivität, damit der Anwender den freundlichen Gruß als Betreff der zu erledigenden Aufgabe erhält.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  ...
  <userTask id="task_hello_world" name="${variables.get('greeting')}"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
  ...
</process>
```

Damit die Prozessaktivitäten in der richtigen Reihenfolge ausgeführt werden, müssen Sie die Sequenzflusselemente noch an die neuen Aktivitäten anpassen.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  ...
  <endEvent id="end" />
  <sequenceFlow id="s1" sourceRef="start" targetRef="call_service" />
  <sequenceFlow id="s2" sourceRef="call_service"
targetRef="receive_service_response" />
  <sequenceFlow id="s3" sourceRef="receive_service_response"
targetRef="task_hello_world" />
  <sequenceFlow id="s4" sourceRef="task_hello_world" targetRef="end" />
</process>
```

Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:di="http://www.omg.org/spec/DD/
20100524/DI"  xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
targetNamespace="" exporter="d.velop process modeler">
  <process id="async_service_process" name="Async Service Process"
```

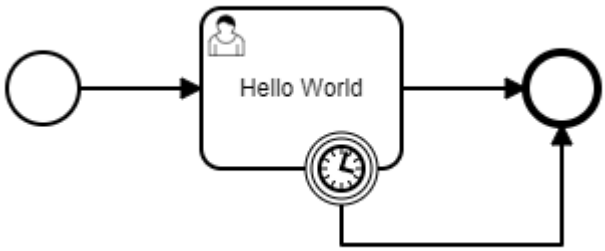
```
isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="service:/process/services/helloworld/
async:in:yourName" value="String" />
        <camunda:property name="service:/process/services/helloworld/
async:out:greeting" value="String" />
        <camunda:property name="service:/process/services/helloworld/
async:name" value="Hello World" />
        <camunda:property name="variable:greeting" value="String" />
      </camunda:properties>
    </extensionElements>
    <startEvent id="start" />
    <sendTask id="call_service"
          name="Call Service 'Hello World'"
          camunda:asyncBefore="true"
          camunda:delegateExpression="${asyncService}"
          camunda:exclusive="true">
      <extensionElements>
        <camunda:inputOutput>
          <camunda:inputParameter name="service.uri">/process/services/
helloworld/async</camunda:inputParameter>
          <camunda:inputParameter name="yourName">$
{variables.getDisplayValue("dv_initiator")}</camunda:inputParameter>
        </camunda:inputOutput>
      </extensionElements>
    </sendTask>
    <receiveTask id="receive_service_response"
          name="Wait for Service 'Hello World'"
          camunda:asyncAfter="true"
          camunda:exclusive="true">
      <extensionElements>
        <camunda:inputOutput>
          <camunda:outputParameter name="greeting">$
{variables.get("greeting")}</camunda:outputParameter>
        </camunda:inputOutput>
      </extensionElements>
    </receiveTask>
    <userTask id="task_hello_world" name="${variables.get('greeting')}"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="call_service" />
    <sequenceFlow id="s2" sourceRef="call_service"
targetRef="receive_service_response" />
    <sequenceFlow id="s3" sourceRef="receive_service_response"
targetRef="task_hello_world" />
    <sequenceFlow id="s4" sourceRef="task_hello_world" targetRef="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="async_service_process">
      <bpmndi:BPMNEdge id="s4_di" bpmnElement="s4">
        <di:waypoint x="630" y="117" />
        <di:waypoint x="662" y="117" />
```

```
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s3_di" bpmnElement="s3">
        <di:waypoint x="490" y="117" />
        <di:waypoint x="530" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="350" y="117" />
        <di:waypoint x="390" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="215" y="117" />
        <di:waypoint x="250" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="179" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="call_service_di" bpmnElement="call_service">
        <dc:Bounds x="250" y="77" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="receive_service_response_di"
bpmnElement="receive_service_response">
        <dc:Bounds x="390" y="77" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="530" y="77" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="662" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

1.4.12. Verwenden von Eskalationen

Beim Modellieren eigener Prozesse können Sie Eskalationen verwenden, um den Prozessablauf zu beeinflussen. Zum Einstieg in das Arbeiten mit Eskalationen ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität und einem Timer-Ereignis erstellen. Dieser Prozess sieht beispielsweise so aus:



Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
```

```
  <process id="escalation_process" name="Escalation Process"
isExecutable="true">
    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <boundaryEvent id="timer" attachedToRef="task_hello_world">
      <timerEventDefinition>
        <timeDuration>PT1M</timeDuration>
      </timerEventDefinition>
    </boundaryEvent>
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
    <sequenceFlow id="s3" sourceRef="timer" targetRef="end" />
  </process>
</definitions>
```

Der Beispielprozess soll nun so angepasst werden, dass beim Eintritt des Timer-Ereignisses eine Eskalation ausgelöst wird. Die Eskalation startet dann einen Subprozess, der den Eskalationspfad beschreibt. Für diese Anpassung erzeugen Sie zunächst ein Eskalationsobjekt.

Hinzufügen eines Eskalationsobjektes

Fügen Sie dem Prozess zunächst das BPMN-Element **escalation** hinzu.

```
...
<definitions ...>
  <process id="escalation_process" name="Escalation Process"
isExecutable="true">
    ...
  </process>
  <escalation id="time_escalation" name="Time Escalation"
escalationCode="123" />
</definitions>
```

Erläuterung der Eigenschaften:

- **id:**Die Eigenschaft wird als eindeutiger Bezeichner für die Eskalation verwendet.
- **name:** Der Anzeigename der Eskalation.
- **escalationCode:** Die Eigenschaft definiert einen technischen Code, der von den auf Eskalationen wartende Elementen verwendet werden kann, um die auslösende Eskalation zu identifizieren.

Nun kann die Eskalation im BPMN-Element **IntermediateThrowEvent** oder einem **EndEvent** verwendet werden, um ein entsprechendes Eskalationsereignis auszulösen.

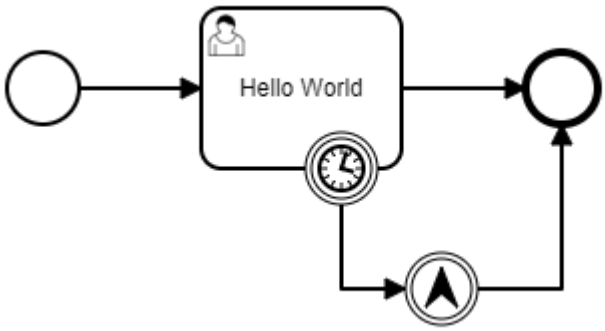
Hinzufügen eines Eskalationsereignisses

Fügen Sie dem Prozess zunächst das BPMN-Element **IntermediateThrowEvent** mit einem **EscalationEventDefinition**-Element hinzu. Anschließend ändern Sie das Ziel des Sequenzflusses **s3** von **end** nach **throw_time_escalation** und fügen einen neuen Sequenzfluss **s4** von **throw_time_escalation** nach **end** hinzu.

```
...
<process id="escalation_process" name="Escalation Process"
isExecutable="true">
  ...
  <intermediateThrowEvent id="throw_time_escalation">
    <escalationEventDefinition escalationRef="time_escalation" />
```

```
  </intermediateThrowEvent>
  <endEvent id="end" />
  ...
  <sequenceFlow id="s3" sourceRef="timer" targetRef="throw_time_escalation"
/>
  <sequenceFlow id="s4" sourceRef="throw_time_escalation" targetRef="end" />
</process>
<escalation id="time_escalation" name="Time Escalation"
escalationCode="123" />
...
```

Der Prozess sieht nun wie folgt aus:



Hinzufügen eines Eskalationspfades

Fügen Sie nun einen Subprozess hinzu, dessen BPMN-Element **StartEvent** das Element **EscalationEvent-Definition** beinhaltet.

```
...
<process id="escalation_process" name="Escalation Process"
isExecutable="true">
  <extensionElements>
    <camunda:properties>
      <camunda:property name="variable:escalationCode" value="String" />
    </camunda:properties>
  </extensionElements>
  <startEvent id="start" />
  ...
  <sequenceFlow id="s4" sourceRef="throw_time_escalation" targetRef="end" />
  <subProcess id="escalation_subprocess" name="Escalation Process"
triggeredByEvent="true">
    <startEvent id="sub_start" isInterrupting="false">
      <escalationEventDefinition escalationRef="time_escalation"
camunda:escalationCodeVariable="escalationCode" />
    </startEvent>
    <endEvent id="sub_end" />
    <sequenceFlow id="sub_s1" sourceRef="sub_start" targetRef="sub_end" />
  </subProcess>
</process>
<escalation id="time_escalation" name="Time Escalation"
escalationCode="123" />
...
```

Erläuterung der Eigenschaften:

- **subProcess**
  - **triggeredByEvent**: muss den Wert **true** enthalten. Die Eigenschaft gibt an, dass das Start-Ereignis dieses Subprozesses durch ein Ereignis ausgelöst wird (in diesem Fall ein Eskalationsereignis).
  - **isInterrupting**: wenn **true**, dann wird der Prozesszweig, der die Eskalation ausgelöst hat, abgebrochen. Wenn **false**, dann läuft dieser Prozesszweig parallel weiter.
- **escalationEventDefinition**
  - **escalationRef:** Enthält die ID der Eskalation, anhand derer dieses Ereignis ausgelöst werden soll.
  - **camunda:escalationCodeVariable:** (Optional) Name der Variablen, in die der Wert von **escalationCode** der aufgetretenen Eskalation gespeichert wird (Variable muss definiert sein).

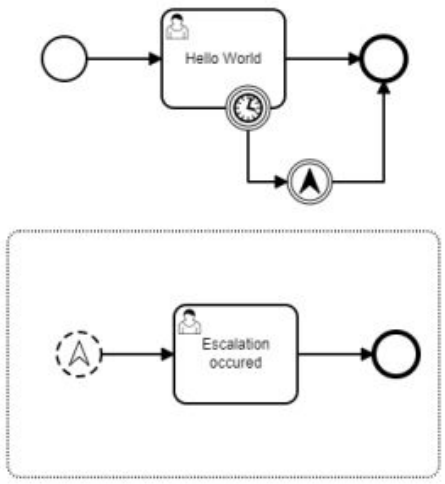
Bitte beachten Sie, dass Sie bei Verwendung der Eigenschaft **camunda:escalationCodeVariable** auch eine Definition der entsprechenden Prozessvariablen im Hauptprozess hinzufügen müssen.

Fügen Sie dem Eskalationspfad nun noch eine Benutzeraktivität hinzu, die im Falle einer Eskalation durchlaufen werden soll.

```
...
<process id="escalation_process" name="Escalation Process"
isExecutable="true">
  ...
  <subProcess id="escalation_subprocess" name="Escalation Process"
triggeredByEvent="true">
    <startEvent id="sub_start" isInterrupting="false">
      <escalationEventDefinition escalationRef="time_escalation"
camunda:escalationCodeVariable="escalationCode" />
    </startEvent>
        <userTask id="sub_task_escalation" name="Escalation occurred"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="sub_end" />
    <sequenceFlow id="sub_s1" sourceRef="sub_start"
targetRef="sub_task_escalation" />
    <sequenceFlow id="sub_s2" sourceRef="sub_task_escalation"
targetRef="sub_end" />
  </subProcess>
</process>
<escalation id="time_escalation" name="Time Escalation"
escalationCode="123" />
...
```

Zur Vereinfachung wird in diesem Beispiel als Empfänger der Benutzeraktivität des Eskalationspfades der Benutzer eingetragen, der den Prozess gestartet hat.

Der finale Prozess sieht wie folgt aus:



Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus.

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:di="http://www.omg.org/spec/DD/
20100524/DI"  xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
targetNamespace="" exporter="d.velop process modeler">
  <process id="escalation_process" name="Escalation Process"
isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="variable:escalationCode" value="String" />
      </camunda:properties>
    </extensionElements>

    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <boundaryEvent id="timer" attachedToRef="task_hello_world">
      <timerEventDefinition>
        <timeDuration>PT1M</timeDuration>
      </timerEventDefinition>
    </boundaryEvent>
    <intermediateThrowEvent id="throw_time_escalation">
      <escalationEventDefinition escalationRef="time_escalation" />
    </intermediateThrowEvent>
    <endEvent id="end" />

    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
    <sequenceFlow id="s3" sourceRef="timer"
targetRef="throw_time_escalation" />
    <sequenceFlow id="s4" sourceRef="throw_time_escalation" targetRef="end"
/>

    <subProcess id="escalation_subprocess" name="Escalation Process"
triggeredByEvent="true">
      <startEvent id="sub_start" isInterrupting="false">
```

```
        <escalationEventDefinition escalationRef="time_escalation"
camunda:escalationCodeVariable="escalationCode" />
      </startEvent>
      <userTask id="sub_task_escalation" name="Escalation occurred"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
      <endEvent id="sub_end" />
      <sequenceFlow id="sub_s1" sourceRef="sub_start"
targetRef="sub_task_escalation" />
      <sequenceFlow id="sub_s2" sourceRef="sub_task_escalation"
targetRef="sub_end" />
    </subProcess>

  </process>
  <escalation id="time_escalation" name="Time Escalation"
escalationCode="123" />

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="escalation_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="179" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="290" y="77" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="timer_di" bpmnElement="timer">
        <dc:Bounds x="372" y="139" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="throw_time_escalation_di"
bpmnElement="throw_time_escalation">
        <dc:Bounds x="432" y="202" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="492" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="215" y="117" />
        <di:waypoint x="290" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="390" y="117" />
        <di:waypoint x="492" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s3_di" bpmnElement="s3">
        <di:waypoint x="390" y="175" />
        <di:waypoint x="390" y="220" />
        <di:waypoint x="432" y="220" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s4_di" bpmnElement="s4">
        <di:waypoint x="468" y="220" />
        <di:waypoint x="510" y="220" />
        <di:waypoint x="510" y="135" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="escalation_subprocess_di"
```

```
bpmnElement="escalation_subprocess" isExpanded="true">
        <dc:Bounds x="165" y="290" width="350" height="200" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="sub_start_di" bpmnElement="sub_start">
        <dc:Bounds x="205" y="372" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="sub_task_escalation_di"
bpmnElement="sub_task_escalation">
        <dc:Bounds x="290" y="350" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="sub_end_di" bpmnElement="sub_end">
        <dc:Bounds x="442" y="372" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="sub_s1_di" bpmnElement="sub_s1">
        <di:waypoint x="241" y="390" />
        <di:waypoint x="290" y="390" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="sub_s2_di" bpmnElement="sub_s2">
        <di:waypoint x="390" y="390" />
        <di:waypoint x="442" y="390" />
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

1.5. Modellieren von Formularen

d.velop forms ist eine Applikation zur Verwaltung von Formularen. Mit der entsprechenden Administratorberechtigung können Sie Formulare erstellen. Anwendende ohne diese Berechtigung können die Formulare öffnen, ausfüllen und absenden. Ein Formular kann dank des reaktionsfähigen Designs auf jedem Endgerät aufgerufen werden.

Jedes erstellte Formular hat eine Einbettungs-URL. Diese URL können Sie verwenden, um Ihren Anwendern das Formular zur Verfügung zu stellen. Sie können das Formular auch innerhalb eines Prozesses verwenden, indem Sie die URL als Bearbeitungsdialog innerhalb eines Benutzerschrittes angeben.

Das Erstellen der Formulare erfolgt mithilfe einer grafischen Benutzeroberfläche. Sie können verschiedene Eingabefelder z. B. für Text, Zahlen, Datumswerte oder eine Datei in Ihrem Formular anordnen. Darüber hinaus haben Sie die Möglichkeit, Ihr Formular mit CSS oder Javascript an Ihre Bedürfnisse anzupassen.

Wenn Anwendende das Formular zu einem Prozessschritt bzw. einem Dokument öffnen, haben Sie innerhalb des Formulars Zugriff auf die Prozessvariablen bzw. Dokumenteigenschaften. Sie können so z. B. eine Zuordnung der Prozessvariablen bzw. Dokumenteigenschaften auf die Formularfelder definieren. Beim Öffnen des Formulars werden die aktuellen Werte der Prozessvariablen bzw. Dokumenteigenschaften automatisch in die Formularfelder geschrieben. Anwendende können diese Werte bearbeiten. Die geänderten Werte werden dann beim Absenden des Formulars wieder in die Prozessvariablen bzw. Dokumenteigenschaften zurückgeschrieben.

1.5.1. Erste Schritte beim Arbeiten mit Formularen

In diesem Thema lernen Sie die Basisfunktionen beim Arbeiten mit Formularen kennen. Sie erfahren, wie Sie Formulare erstellen, öffnen und verwalten.

Aufrufen eines Formulars

Damit andere Benutzer Ihr Formular öffnen, ausfüllen und absenden können, brauchen Sie die URL des Formulars. Diese URL wird in der Adresszeile des Browsers angezeigt, wenn Sie das Formular als

Vorschau öffnen. Wenn Sie auf **Vorschau** klicken, öffnet sich das Formular in einem neuen Browsertab. Dies kann nützlich sein, um das Formular zu testen.

Alternativ erhalten Sie die URL eines Formulars, indem Sie in der Formularliste im Dreipunktemenü des entsprechenden Formulars **URL in die Zwischenablage kopieren** auswählen. Nun können Sie die URL aus der Zwischenablage entweder direkt an andere Benutzer verteilen oder Sie verwenden die URL, um das Formular in einer anderen Anwendung (wie z.B. in einer Aufgabe oder einem Prozess) einzubinden.

> **Anmerkung**
> Beim Kopieren in die Zwischenablage wird die absolute URL verwendet. Beim Einbinden der URL in andere Anwendungen sollten Sie die URL relativ angeben.

Erstellen eines Formulars

Sie können ein neues Formular in der Formularverwaltung erstellen. Voraussetzung ist, dass Sie über die entsprechenden Berechtigungen verfügen.

So geht's

1. Klicken Sie auf die Schaltfläche zum Hinzufügen eines Formulars (Plussymbol) in der Toolbar.
2. Geben Sie einen Namen für das Formular ein und drücken Sie auf **Erstellen**.
3. Wählen Sie eine Komponente aus dem Komponentenmenü aus und ziehen Sie sie per Drag & Drop in Ihr Formular. Beschränken Sie sich für den Anfang am besten auf die Komponenten der Kategorie **Basic**.
4. Passen Sie das Label der Komponente an. Die übrigen Einstellungen können Sie später ausprobieren.
5. Klicken Sie auf **Save**.
6. Fügen Sie bei Bedarf weitere Komponenten hinzu.
7. Speichern Sie Ihr Formular.

> **Anmerkung**
> Als Formular-Builder verwendet d.velop forms die Komponente "Form Builder" von form.io . Eine Anleitung zur Verwendung dieser Komponente finden Sie hier.

Importieren und Exportieren von Formularen

Sie haben die Möglichkeit, Formulare zu exportieren und zu importieren. Das ist beispielsweise hilfreich, wenn Sie ein Formular von einem System in ein anderes übertragen möchten.

So geht's

1. Öffnen Sie die Formularübersicht im Quellsystem, z.B. indem Sie das Feature **Prozessstudio** auf der Startseite öffnen.
2. Öffnen Sie das Sie das Dreipunktemenü neben dem entsprechenden Formular und wählen Sie **Formular herunterladen** aus. Die Formulardefinition wird nun als JSON-Datei heruntergeladen.
3. Öffnen Sie die Formularübersicht im Zielsystem.
4. Klicken Sie in der Toolbar auf den Importbutton (⬆) und wählen Sie Ihr exportiertes Formular aus.

Verwalten der Formulare

In der Konfiguration finden Sie unter **Aufgaben und Prozesse** den Bereich **Formulare**. Mit dem Eintrag **Formularverwaltung** navigieren Sie zur Übersichtsliste aller vorhandenen Formulare. In dieser Übersichtsliste können Sie neue Formulare erstellen sowie bestehende Formulare bearbeiten oder löschen.

> **Anmerkung**
> Sie können den Eintrag **Formularverwaltung** nur dann sehen, wenn Ihnen die Benutzerrolle **Administration** zugewiesen ist. Diese kann Ihnen nur Ihre Administration zuweisen.

1.5.2. Formulare innerhalb von Benutzeraktivitäten

Sie können Benutzeraufgaben mithilfe eines Formulars abschließen. Um eine Aufgabe aus dem Formular heraus abzuschließen, haben Sie drei Möglichkeiten:

1. Verwenden einer "Event"-Schaltfläche im Formular
2. Verwenden einer benutzerspezifischen Aktion in einem Skript
3. Verwenden der Methode **window.postMessage** in einem Skript

Abschließen der Aufgabe mit einer "Event"-Schaltfläche - So geht's

1. Fügen Sie Ihrem Formular eine Schaltfläche hinzu.
2. Konfigurieren Sie die Schaltfläche wie folgt:
   - **Action: Event**
   - **Button Event: taskCompletion**

Wenn ein Anwender die Aufgabe mit dem Formular öffnet und auf die Schaltfläche klickt, wird die Aufgabe abgeschlossen. Zusätzlich werden evtl. vorhandene Prozessvariablen und Dokumenteigenschaften gespeichert.

Abschließen der Aufgabe mit einem Event mit einer benutzerspezifischen Aktion - So geht's

1. Fügen Sie Ihrem Formular eine Komponente hinzu, die die Ausführung eines Skripts unterstützt, z. B. eine Schaltfläche mit der Action **Custom**.
2. Rufen Sie in dem Skript folgenden Code auf:

```
form.emit("customEvent", {"type":"taskCompletion"});
```

Wenn ein Anwender die Aufgabe mit dem Formular öffnet und auf die Schaltfläche klickt, wird die Aufgabe abgeschlossen. Zusätzlich werden evtl. vorhandene Prozessvariablen und Dokumenteigenschaften gespeichert.

Anders als beim Abschließen der Aufgabe mit einer "Event"-Schaltfläche haben Sie bei dieser Variante die Möglichkeit, vor dem Aufgabenabschluss noch weiteren Custom-Code auszuführen.

Abschließen der Aufgabe mithilfe der Methode "parent.postMessage" - So geht's

1. Fügen Sie Ihrem Formular eine Komponente hinzu, welche die Ausführung eines Skripts unterstützt, z. B. einen Button mit der Aktion **Custom**.
2. Rufen Sie in dem Skript folgenden Code auf:

```
parent.postMessage("TaskApp.completeTask", "*");
```

Dies schließt die Aufgabe ab, ohne evtl. vorhandene Prozessvariablen und Dokumenteigenschaften zu speichern.

1.5.3. Verwenden von Dokumenteigenschaften im Formular

Um in einem Formular Zugriff auf die Eigenschaften eines Dokuments zu haben, können Sie die Formular-URL mit den folgenden Request-Parametern aufrufen:

- **dmsRepoId=<dms-repo-id>**
- **dmsObjectId=<dms-object-id>**

Die Dokumenteigenschaften werden automatisch beim Laden des Formulars aus der DMS-App ermittelt und im Formular mit der Variable **dmsProperties** als JSON-Objekt zur Verfügung gestellt.

Beispiel für das JSON-Objekt "dmsProperties"

```
{
  "myProperty" : "Hello!",
```

```
  "otherProperty" : 4711
}
```

> **Anmerkung**
> Wenn der Key einer Dokumenteigenschaft Punkte enthält, werden daraus Unterobjekte.
>
> **Beispiel:**
>
> Aus der Dokumenteigenschaft mit dem Key **this.is.a.key** und dem Wert **My value** wird folgendes JSON-Objekt:
>
> ```
> {
>   "this" : {
>     "is" : {
>       "a" : {
>         "key" : "My value"
>       }
>     }
>   }
> }
> ```

Sie können die Dokumenteigenschaften auf zwei Arten verwenden:

1. In einem Skript
2. In einem Formularelement

Verwenden einer Dokumenteigenschaft in einem Skript

Sie können in einem Skript jederzeit sowohl lesend als auch schreibend auf das JSON-Objekt zugreifen. Wenn das Formular am Ende mit dem Event **taskCompletion** oder **saveDmsProperties** abgeschickt wird, werden die aktuellen Werte aus dem JSON-Objekt ermittelt und in die Dokumenteigenschaften geschrieben.

Binden eines Formularelements an eine Dokumenteigenschaft

Um ein Formularelement an eine Dokumenteigenschaft zu binden, öffnen Sie in den Einstellungen des Formularelements die Registerkarte **API** und geben Sie die Dokumenteigenschaft in der Form **dmsProperties.<´dms-property-key>** unter **Property Name** an. Dieses Formularelement wird nun beim Laden des Formulars mit dem Wert der Dokumenteigenschaft vorausgefüllt. Der Benutzer, welcher das Formular geöffnet hat, kann den Wert danach ggf. anpassen. Wenn das Formular am Ende mit dem Event **taskCompletion** abgesendet wird, wird der aktuelle Wert des Formularelements wieder in die Dokumenteigenschaften geschrieben.

> **Warnung**
> Die Dokumenteigenschaften werden im Namen des angemeldeten Benutzers, welcher das Formular geöffnet hat, geladen und gespeichert. Stellen Sie sicher, dass der Benutzer Berechtigungen für das Dokument hat.

1.5.4. Verwenden von Prozessvariablen im Formular

Sie können ein Formular als Bearbeitungsansicht einer Benutzeraufgabe in einem BPMN-Prozess verwenden, der mit d.velop process ausgeführt wird. Geben Sie hierzu die URL des Formulars (möglichst als relative URL) als **Form Key** der Benutzeraufgabe in der Form "uri:<form-url>" an.

Damit das Formular auf Prozessvariablen zugreifen kann, fügen Sie der URL den Request-Parameter **processVariablesUri=${process.task.variablesUri}** hinzu.

Die Prozessvariablen werden automatisch beim Laden des Formulars aus dem Prozess ermittelt und im Formular mit der Variable **processVariables** als JSON-Objekt zur Verfügung gestellt.

Beispiel für das JSON-Objekt "processVariables"

```
{
  "myVariable" : "Hello!",
  "otherVariable" : 4711
}
```

Die Prozessvariablen können auf zwei Arten verwendet werden:

1. In einem Skript
2. In einem Formularelement

Verwenden einer Prozessvariable in einem Skript

Sie können in einem Skript jederzeit sowohl lesend als auch schreibend auf das JSON-Objekt zugreifen. Wenn das Formular am Ende mit dem Event **taskCompletion** abgesendet wird, werden die aktuellen Werte für die Prozessvariablen aus dem JSON-Objekt ermittelt und an den Prozess gesendet.

Binden eines Formularelements an eine Prozessvariable

Um ein Formularelement an eine Prozessvariable zu binden, öffnen Sie in den Einstellungen des Formularelements die Registerkarte **API** und geben Sie die Prozessvariable in der Form **processVariables.<process-variable-key>** unter **Property Name** an. Dieses Formularelement wird nun beim Laden des Formulars mit dem Wert der Prozessvariablen vorausgefüllt. Der Benutzer, welcher das Formular geöffnet hat, kann den Wert danach ggf. anpassen. Wenn das Formular am Ende mit dem Event **taskCompletion** oder **saveProcessVariables** abgesendet wird, wird der aktuelle Wert des Formularelements wieder als Prozessvariable an den Prozess übermittelt.

> **Anmerkung**
> Die Prozessvariablen werden im Namen des angemeldeten Benutzers geladen und gespeichert. Damit dieser das Recht dazu hat, muss er die zugehörige Aufgabe aktuell in seiner Bearbeitung haben. Dies ist dadurch gegeben, dass das Formular in der Aufgabe angezeigt wird.

1.5.5. Verwenden eines Formulars in einer Aufgabe

Sie können ein Formular als Bearbeitungsansicht in einer Aufgabe verwenden, indem Sie die URL des Formulars beim Erstellen der Aufgabe mit der API der Task-App als Link **form** mitgeben. Wir empfehlen, die URL relativ anzugeben.

In einigen Fällen werden die Kontextaktionen **Erledigen**, **Weiterleiten** und **Übernehmen** für Aufgaben nicht benötigt, wenn ein Formular als Bearbeitungsansicht verwendet wird. Sie können die Kontextaktionen in der Aufgabe ausblenden. Fügen Sie zum Ausblenden der Formular-URL die entsprechenden folgenden Requestparameter hinzu:

- **disableTaskAction=complete**
- **disableTaskAction=forward**
- **disableTaskAction=claim**

Abschließen einer Aufgabe aus einem Formular heraus

Sie können Benutzeraufgaben mithilfe eines Formulars abschließen. Um eine Aufgabe aus dem Formular heraus abzuschließen, haben Sie drei Möglichkeiten:

1. Verwenden einer "Event"-Schaltfläche im Formular

2. Verwenden einer benutzerspezifischen Aktion in einem Skript
3. Verwenden der Methode **window.postMessage** in einem Skript

Abschließen der Aufgabe mit einer "Event"-Schaltfläche - So geht's

1. Fügen Sie Ihrem Formular eine Schaltfläche hinzu.
2. Konfigurieren Sie die Schaltfläche wie folgt:
   - **Action: Event**
   - **Button Event: taskCompletion**

Wenn ein Anwender die Aufgabe mit dem Formular öffnet und auf die Schaltfläche klickt, wird die Aufgabe abgeschlossen. Zusätzlich werden evtl. vorhandene Prozessvariablen und Dokumenteigenschaften gespeichert.

Abschließen der Aufgabe mit einem Event mit einer benutzerspezifischen Aktion - So geht's

1. Fügen Sie Ihrem Formular eine Komponente hinzu, die die Ausführung eines Skripts unterstützt, z. B. eine Schaltfläche mit der Action **Custom**.
2. Rufen Sie in dem Skript folgenden Code auf:

```
form.emit("customEvent", {"type":"taskCompletion"});
```

Wenn ein Anwender die Aufgabe mit dem Formular öffnet und auf die Schaltfläche klickt, wird die Aufgabe abgeschlossen. Zusätzlich werden evtl. vorhandene Prozessvariablen und Dokumenteigenschaften gespeichert.

Anders als beim Abschließen der Aufgabe mit einer "Event"-Schaltfläche haben Sie bei dieser Variante die Möglichkeit, vor dem Aufgabenabschluss noch weiteren Custom-Code auszuführen.

Abschließen der Aufgabe mithilfe der Methode "parent.postMessage" - So geht's

1. Fügen Sie Ihrem Formular eine Komponente hinzu, welche die Ausführung eines Skripts unterstützt, z. B. einen Button mit der Aktion **Custom**.
2. Rufen Sie in dem Skript folgenden Code auf:

```
parent.postMessage("TaskApp.completeTask", "*");
```

Dies schließt die Aufgabe ab, ohne evtl. vorhandene Prozessvariablen und Dokumenteigenschaften zu speichern.

Speichern von Dokumenteigenschaften und Prozessvariablen mithilfe eines Formulars

Sie können Dokumenteigenschaften und Prozessvariablen speichern, ohne eine Aufgabe abzuschließen. Sie haben für das Speichern ohne Abschließen der Aufgabe zwei Möglichkeiten:

1. Verwenden einer "Event"-Schaltfläche
2. Verwenden einer benutzerspezifischen Aktion

Speichern der Dokumenteigenschaften und Prozessvariablen mit einer "Event"-Schaltfläche - So geht's

So geht's

1. Fügen Sie Ihrem Formular eine Schaltfläche hinzu.
2. Konfigurieren Sie die Schaltfläche wie folgt:
   - **Action: Event**
   - **Button Event: save**

Beim Klick auf die Schaltfläche werden evtl. vorhandene Prozessvariablen und Dokumenteigenschaften gespeichert. Wenn keine Prozessvariablen oder Dokumenteigenschaften vorhanden sind, wird die Speicherung übersprungen.

Speichern der Dokumenteigenschaften und Prozessvariablen mit einem Event einer benutzerspezifischen Aktion -So geht's

1. Fügen Sie Ihrem Formular eine Komponente hinzu, die die Ausführung eines Skripts unterstützt, z. B. eine Schaltfläche mit der Action **Custom**.
2. Rufen Sie in dem Skript folgenden Code auf:

```
form.emit("customEvent", {"type":"save"});
```

Öffnet ein Anwender die Aufgabe mit dem Formular und klickt auf die Schaltfläche werden evtl. vorhandene Prozessvariablen und Dokumenteigenschaften gespeichert. Wenn keine Prozessvariablen oder Dokumenteigenschaften vorhanden sind, wird die Speicherung übersprungen.

Anders als beim Speichern mit einer "Event"-Schaltfläche haben Sie bei dieser Variante die Möglichkeit, vor dem Speichervorgang noch weiteren benutzerspezifischen Code auszuführen.

Mit dem Typ **saveDmsProperties** können Sie Dokumenteigenschaften speichern, ohne dass gleichzeitig Prozessvariablen gespeichert werden:

```
form.emit("customEvent", {"type":"saveDmsProperties"});
```

Mit dem Typ **saveProcessVariables** können Sie Prozessvariablen speichern, ohne dass gleichzeitig DMS-Objekteigenschaften gespeichert werden:

```
form.emit("customEvent", {"type":"saveProcessVariables"});
```

Sie können folgendermaßen auf das Ergebnis der verschiedenen Speicherevents reagieren:

```
form.on("saveSuccess", () => { console.log("Saved successfully") });
form.on("saveFailed", () => { console.error("Failed to save") });
```

1.6. Verwenden von Skripten

Mithilfe von d.velop scripting können Sie Skripte erstellen und verwalten. Mit Skripten können Sie Ihre Umgebung mit kundenspezifischen Logiken erweitern: Binden Sie die erstellten Skripte im Rahmen von z.B. Prozessen oder Webhooks (d.velop documents) ein.

Mit der entsprechenden Berechtigung können Sie Skripte erstellen und bearbeiten. Anwendende ohne diese Berechtigung können Skripte ausführen.

Jedes erstellte Skript hat eine Ausführungs-URL. Diese URL können Sie verwenden, um das jeweilige Skript mittels eines **GET-**, **POST-**, **PATCH-**, **PUT-** oder **DELETE**-Requests auszuführen.

1.6.1. Erste Schritte beim Arbeiten mit Skripten

In diesem Thema lernen Sie die Basisfunktionen beim Arbeiten mit Skripten kennen. Sie erfahren, wie Sie Skripte erstellen öffnen und verwalten.

Aufrufen eines Skriptes

Damit ein Skript ausgeführt werden kann, brauchen Sie die URL des Skriptes. Sie erhalten die URL eines Skriptes, indem Sie in der Skriptliste im Dreipunktemenü des entsprechenden Skripts **URL in die Zwischenablage kopieren** auswählen. Sie können diese URL nun verwenden, um das Skript auszuführen.

Sie können das Skript mit den folgenden HTTP-Methoden aufrufen:

- **GET**

- **POST**
- **PUT**
- **PATCH**
- **DELETE**

Bei den Methoden **GET** und **DELETE** ist es nicht möglich, Daten im Requestbody an das Skript zu übergeben.

> **Anmerkung**
> Ein Skript kann nur von einem Benutzers des Systems ausgeführt werden.

Erstellen eines Skriptes

Sie können ein neues Skript in der Skriptverwaltung erstellen. Voraussetzung ist, dass Sie über die entsprechenden Berechtigungen verfügen.

So geht's

1. Klicken Sie auf die Schaltfläche zum Hinzufügen eines Skriptes in der Toolbar (Plussymbol).
2. Geben Sie einen Namen für das Skript ein und klicken Sie auf **Erstellen.** Im Editorbereich wird Ihnen bereits der Funktionsrahmen eines lauffähigen "Hallo Welt"-Skriptes angezeigt, welches Sie anpassen oder erweitern können.
3. Speichern Sie Ihr Skript.

Importieren und Exportieren von Skripten

Sie haben die Möglichkeit, Skripte zu exportieren und zu importieren. Das ist beispielsweise hilfreich, wenn Sie ein Skript von einem System in ein anderes übertragen möchten.

So geht's

1. Öffnen Sie die Skriptübersicht im Quellsystem, z.B. indem Sie das Feature **Prozessstudio** auf der Startseite öffnen.
2. Öffnen Sie das Dreipunktemenü neben dem entsprechenden Skript und wählen Sie **Skript herunterladen** aus. Die Skriptdefinition wird nun als JSON-Datei heruntergeladen.
3. Öffnen Sie die Skriptübersicht im Zielsystem.
4. Klicken Sie in der Toolbar auf die Importschaltfläche ( ) und wählen Sie Ihr exportiertes Skript aus.

> **Anmerkung**
> Verschlüsselte Variablen werden mit einem leeren Wert exportiert. Nach dem Import müssen Sie diese Variablen ggf. nachpflegen. Sie können entscheiden, ob Variablen auch importiert werden sollen oder nicht. Wenn die Variablen auch importiert werden sollen, werden alle bereits vorhandenen Variablen beim Überschreiben eines Skriptes durch den Import überschrieben.

Verwalten von Skripten

In der Konfiguration finden Sie unter **Aufgaben und Prozesse** den Bereich **Skripting**. Mit dem Eintrag **Skriptverwaltung** navigieren Sie zur Übersichtsliste aller vorhandenen Skripte. In der Übersichtsliste können Sie neue Skripte erstellen sowie bestehende Skripte bearbeiten oder löschen.

> **Anmerkung**
> Sie können den Eintrag **Skriptverwaltung** nur dann sehen, wenn Ihnen die Benutzerrolle **Administration** zugewiesen ist. Diese kann Ihnen nur Ihre Administration zuweisen.

1.6.2. Verwenden von externen Bibliotheken in einem Skript

Innerhalb eines Skriptes können Sie externe Bibliotheken verwenden, um z.B. Drittsysteme anzusprechen.

So geht's

1. Wechseln Sie zur Registerkarte **Externe Bibliotheken**, wenn Sie eine externe Bibliothek verwenden möchten.
2. Fügen Sie mit dem Plussymbol eine neue externe Bibliothek hinzu. Sie können Pakete verwenden, die bei npmjs.com angeboten werden.
3. Geben Sie einen gültigen Bibliotheksnamen an und klicken Sie auf **Übernehmen**. Die angegebene Bibliothek wird validiert. Wenn Sie keine explizite Version definieren, wird automatisch die letzte Versionsnummer übernommen.
4. Importieren Sie die Bibliothek mit der Funktion **require** ins Skript. Die Bibliothek kann nachfolgend im Skriptcode verwendet werden.

> **Anmerkung**
> Beim Hinzufügen der externen Bibliothek wird diese von npmjs.com heruntergeladen. Um diese Funktionalität verwenden zu können, muss d.velop scripting die Adresse npmjs.com im Internet erreichen können.

1.6.3. Verwenden von Variablen

Sie können Variablen erstellen und innerhalb eines Skripts verwenden. Variablen sind beispielsweise hilfreich, wenn Sie keine API-Schlüssel oder Kennwörter im Code eintragen möchten.

So geht's

1. Öffnen Sie das entsprechende Skript.
2. Navigieren Sie zu **Variablen** und klicken Sie auf **Variable hinzufügen**.
3. Tragen Sie unter **Name** einen Namen für die Variable ein. Mit dem Namen können Sie die Variable innerhalb des Skriptes referenzieren.
4. Tragen Sie unter **Wert** einen Wert für die Variable ein. Aktivieren Sie optional **verschlüsselt**, um den Wert verschlüsselt zu speichern.
5. Verwenden Sie das **Request**-Objekt, um die erstellte Variable zu referenzieren.

1.6.4. Testen eines erstellten Skriptes

Sie können Ihre erstellten Skripte jederzeit testen, um die Lauffähigkeit zu überprüfen.

So geht's

1. Wählen Sie das entsprechende Skript aus der Skriptliste aus.
2. Klicken Sie auf **Testen**, um den Testbereich einzublenden.
3. Klicken Sie auf **Durchführen**, um das zugehörige Skript aufzurufen. Die Skriptantwort (Response) sowie der dazugehörige Statuscode werden angezeigt.

1.6.5. Arbeiten mit dem "Request"-Objekt

Wenn Sie ein Skript erstellen, können Sie innerhalb des Skriptes ein **Request**-Objekt verwenden. Mithilfe des **Request**-Objektes können Sie Variablen referenzieren, die Sie vorab erstellt haben. Verwenden Sie dazu die Funktionen **var(name)** und **variables()**.

Sie können je nach Format mittels **data()**, **json()** oder **text()** auf den Request-Body zugreifen.

Verwenden Sie zum Zugreifen auf Anfrageparameter die Funktion **param(name)**.

Wenn Sie einen bestimmten Header-Wert auslesen möchten, können Sie dazu die Funktion **get(name)** verwenden.

Referenz:

```
class Request {
    /**
     * @type {Object.<string, string>}
     */
    #headers;
    /**
     * @type {Buffer}
     */
    #body;
    /**
     * @type {Object.<string, string>}
     */
    #variables;

    constructor(method, path, params, headers, body, variables) {
        this.method = method;
        this.path = path;
        this.params = params;
        this.#headers = headers;
        this.#body = body;
        this.#variables = variables;
    }

    get(name) {
        if (!name) {
            return this.#headers;
        }
        return this.#headers[name.toLowerCase()];
    }

    param(name) {
        if (!name) {
            return this.params;
        }
        return this.params[name.toLowerCase()];
    }

    var(name) {
        return this.#variables[name]
    }

    variables() {
        return this.#variables;
    }

    data() {
        return this.#body;
    }

    end() {
        console.warn("req.end() is deprecated and will be removed in a
```

```
future version!")
    }

    json() {
        return JSON.parse(this.#body.toString('utf-8'));
    }

    text() {
        return this.#body.toString('utf-8');
    }

}
```

1.6.6. Arbeiten mit dem "Response"-Objekt

Wenn Sie ein Skript erstellen, können Sie innerhalb des Skriptes ein **Response**-Objekt verwenden.

Das **Response**-Objekt ist das Ergebnis des Skripts. Sie können den Statuscode, den Response Body und beliebige Header-Werte übermitteln.

Zum Übermitteln des HTTP-Statuscodes können Sie die **status(code)**-Methode verwenden.

Wenn Sie Ihre Antwort im JSON-Format geben möchten, verwenden Sie die Methode **json(obj)**. Für andere Formate können Sie die Methode **send(body)** verwenden und den passenden **Content-Type**-Header mittels **set(key, value)** einfügen.

Die **set(key,value)**-Methode ermöglicht auch das Übermitteln beliebiger weiterer Header-Werte. Der Parameter **key** ist der Headername und **value** ist der entsprechende Wert.

Das **Response**-Objekt sieht wie folgt aus:

```
class Response {
    #header = {};
    #status = 0;
    #body = Buffer.alloc(0);

    status(code) {
        this.#status = code;
        return this;
    }

    json(obj) {
        if (Buffer.isBuffer(obj)) {
            this.set("Content-Type", "application/
json").send(JSON.stringify(obj)); // If a buffer gets here explicitly
            return;
        }
        this.set("Content-Type", "application/json").send(obj);
    }

    set(key, value) {
        if (value === undefined) {
            if (typeof key === 'object') {
                this.#header = key;
                return this;
            }
```

```
            throw new Error('when value is left out then key needs to be a
map of strings');
        }
        if (this.#header === undefined || this.#header === null) {
            this.#header = {};
        }
        this.#header[key] = value;
        return this;
    }

    send(body) {
        if (typeof body === 'string') {
            this.#body = Buffer.from(body, 'utf-8');
        } else if (Buffer.isBuffer(body)) {
            this.#body = body;
        } else if (typeof body === 'object') {
            this.#body = Buffer.from(JSON.stringify(body), 'utf-8');
        } else
            console.error("Body is not acceptable!", typeof body)
    }

    body() {
        return this.#body
    }

    header() {
        return this.#header
    }

    //Returns HTTP Status Code
    getStatus() {
        return this.#status
    }

    //Deprecated
    end() {
        console.warn("res.end() is deprecated and will be removed in a
future version!")
    }

}
```

Aktivieren des Offline-Modus von d.velop scripting für Systeme ohne Internetzugang

Ab Version 1.3.0 bzw. 2023.Q4 erkennt d.velop scripting automatisch, wenn kein Internetzugang auf dem Server besteht. Wenn Sie d.velop scripting in der On-Premises-Bereitstellung verwenden und auf dem Server kein Internetzugang besteht, können Sie eine Umgebungsvariable hinzufügen. Die Umgebungsvariable überspringt die automatische Erkennung des Internetzugangs und schaltet d.velop scripting permanent offline.

So geht's

1. Öffnen Sie mit **Windows-Taste + R** den **Ausführen**-Dialog von Windows.
2. Geben Sie **sysdm.cpl** ein und bestätigen Sie mit **OK**.
3. Navigieren Sie zu **Erweitert > Umgebungsvariablen**.

4. Erstellen Sie unter **Systemvariablen** die Variable **node.offline** mit dem Wert **true**.
5. Starten Sie d.3 process manager neu.

Sie haben den Offline-Modus von d.velop scripting aktiviert. Beachten Sie folgende Hinweise zum Offline-Modus:

- Beim Erstellen von Skripten wird **npm install** nicht mehr ausgeführt.
- Bibliotheken, die über d.velop scripting hinzugefügt wurden, werden nicht mehr berücksichtigt. Kopieren Sie eventuell benötigte Bibliotheken manuell in das Verzeichnis C:\d3\nodejs18\node_modules.
- d.velop scripting versucht zunächst, den Ordner **node_modules** zu verknüpfen. Wenn Berechtigungen für die Verknüpfung fehlen, wird der Ordner für die Ausführung kopiert. Wenn der Ordner nicht verknüpft oder kopiert werden kann, fügen Sie den folgenden Code an den Anfang des Skripts hinzu:

```
process.env.NODE_PATH = "C:\\d3\\nodejs18\\node_modules"
require("module").Module._initPaths();
```

1.7. Verwenden von Ereignissen mit Dokumentenbezug

In d.velop process studio können Sie Ereignisse mit Bezug zu Dokumenten bereitstellen. Diese Ereignisse können Sie z.B. konsumieren, um Prozesse zu starten. Weitere Informationen erhalten Sie in der Anwendendendokumentation von d.velop process studio.

Folgende Ereignisse stehen zur Verfügung:

- Nach dem Erstellen eines Dokuments oder einer Akte
- Nach dem Erstellen einer neuen Dokumentversion
- Nach dem Aktualisieren von Dokumenteigenschaften
- Nach der Änderung eines Dokumentenstatus
- Nach der Freigabe eines Dokuments
- Nach dem Löschen eines Dokuments oder einer Akte
- Nach dem Erstellen eines generierten Dokuments

Die Bereitstellung dieser Ereignisse muss durch ein Mitglied der technischen Administration konfiguriert werden. Initial ist eine Synchronisierung mit d.velop documents notwendig, damit die verfügbaren Daten aus dem Repository (wie z.B. Kategorien) zur Verfügung stehen. Wenn mehrere Repositorys konfiguriert sind, wird entweder das Standardrepository oder (wenn kein Standardrepository konfiguriert wurde) das erste Repository zu Grunde gelegt.

So geht's

1. Wählen Sie das Feature **Konfiguration** aus.
2. Wählen Sie im Bereich **Ereignisse** die Option **Repository konfigurieren** aus.
3. Stellen Sie anhand der einleitenden Beschreibung sicher, dass das korrekte Repository ermittelt wurde.
4. Klicken Sie auf **Synchronisieren**. Die benötigten Daten aus dem Repository werden mit den Berechtigungen des angemeldeten Benutzers ermittelt. Wenn sich die relevanten Berechtigungen oder Daten (wie z.B. Kategorien oder Eigenschaften) geändert haben, wiederholen Sie diesen Vorgang.
5. Aktivieren Sie **Zugriff auf erweiterte Eigenschaften erlauben**, wenn die Eigenschaften der Dokumente an die Konsumierenden der Ereignisse übermittelt werden sollen.
6. Tragen Sie (sofern zutreffend) unter **Servicebenutzer aus d.velop documents eintragen** die ID des Benutzers ein, der durch den in d.velop documents eingetragenen API-Schlüssel repräsentiert wird und in dessen Namen Webhooks versendet werden.
7. Aktivieren Sie alle Ereignisse aus d.velop documents, die zum Konsumieren angeboten werden sollen.

1.8. Administrieren von Prozessen

In diesem Thema erfahren Sie, wie Sie Ihre Prozesse verwalten können. Informieren Sie sich über die Möglichkeiten, einen Überblick über Ihre Prozesse zu erhalten und fehlerhafte Prozesse zu korrigieren.

1.8.1. Wissenswertes zum Feature "Prozessadministration"

Mit dem Feature **Prozessadministration** können Sie Aktionen starten, die ganze Prozessversionen betreffen. In der Regel umfassen die Aktionen mehrere Prozessinstanzen. Die Prozessadministration erlaubt es Ihnen, Aktionen zu verschiedenen Prozessen und Versionen nacheinander durchzuführen, ohne die Ansicht wechseln zu müssen.

Sie können folgende Aktionen für ganze Prozessversionen durchführen:

- **Prozessversionen erneut ausführen**: Wenn innerhalb eines Prozesses mehrere Fehler auftreten, können Sie zur Fehlerbehebung die Prozessversion erneut ausführen. Verwenden Sie zum erneuten Ausführen einer Prozessversion die Aktion **Erneut ausführen**.
- **Prozessinstanzen in die nächste Prozessversion migrieren**: Bei der Aktualisierung von Prozessen wird automatisch eine neue Version des Prozesses angelegt. Sie können mit der Aktion **Migrieren** alle Instanzen einer Prozessversion in die neuste Version migrieren.
- **Prozessversionen löschen**: Wenn keine laufenden Prozessinstanzen zu einer Version mehr vorliegen, können Sie die Prozessversion löschen. Verwenden Sie zum Löschen einer Prozessversion die Aktion **Prozessversion löschen**.
- **Prozessinstanzen abbrechen**: Sie können alle offenen Instanzen einer Prozessversion abbrechen. Durch das Abbrechen werden alle offenen Aktivitäten zu dieser Prozessversion entfernt und die Prozessinstanz kann nicht wieder fortgesetzt werden. Verwenden Sie zum Abbrechen einer Prozessinstanz die Aktion **Instanzen abbrechen**.

Darüber hinaus stehen noch folgende Aktionen zur Verfügung:

- **Neuen Prozess bereitstellen**: Sie können einen neuen Prozess (oder eine neue Version eines bereits bestehenden Prozesses) aus einer **.bpmn**-Datei ins System importieren.

1.8.2. Anzeigen aller Prozessinstanzen

Mit dem Feature **Prozessüberwachung** erhalten Sie einen Überblick über alle Prozessinstanzen. Sie können mit dem Business-Key gezielt nach einzelnen Prozessen suchen oder die Ergebnisliste mit den Filtern **Prozesse**, **Versionen**, **Aktivitäten** und **Status** eingrenzen. Ein Prozess befindet sich in einem der folgenden vier Status:

 : Die Instanz wurde abgebrochen.

 : Die Instanz ist in einem Fehlerzustand.

 : Die Instanz wurde erfolgreich beendet.

 : Die Instanz läuft aktuell noch.

Ein zusätzliches Ausrufungszeichen beim Status signalisiert einen Fehler oder eine Warnung beim Prozessprotokollexport.

Um nähere Informationen zu einer Prozessinstanz zu erhalten, wählen Sie den Prozess aus. In der Detailansicht stehen Ihnen Kontextaktionen zur Verfügung, die sich je nach Prozesstyp unterscheiden können. Mit den Kontextaktionen können Sie die Prozessinstanz abbrechen, in die nächste Version migrieren, in einem Instanzdiagramm darstellen oder in der Quelle anzeigen. Für eine Übersicht über den Prozessverlauf können Sie außerdem das Prozessprotokoll anzeigen.

1.8.3. Anzeigen eines Instanzdiagramms

In der Detailansicht einer Instanz können Sie die Prozessinstanz mit der Kontextaktion **Instanzdiagramm** in einem Diagramm anzeigen.

Um bei sehr komplexen Instanzdiagrammen den Überblick zu bewahren, können Sie die Ansicht des Diagramms ändern. Mit dem Mausrad können Sie die Größe des Diagramms ändern. Wenn Sie die Ansicht mit dem Mausrad vergrößert haben, können Sie das Diagramm mit gedrückter linker Maustaste verschieben. Mit **Ansicht zurücksetzen** kehren Sie zur ursprünglichen Ansicht zurück.

1.8.4. Ändern der Variablen eines Tokens

Wenn Sie im Feature **Prozessüberwachung** in die Detailansicht eines Tokens navigieren, können Sie die Variablen der Prozessinstanz ändern. Ändern Sie Variablen beispielsweise, um die mögliche Fehlerursache eines Prozesses zu beheben.

Wählen Sie **Variablen bearbeiten** aus, um den Bearbeitungsmodus zu aktivieren und die Werte in den Eingabefeldern zu ändern. Sie haben zwei Möglichkeiten, den Bearbeitungsmodus zu verlassen:

- Wählen Sie **Änderungen speichern** aus, um die neuen Werte in die Datenbank zu schreiben und den Bearbeitungsmodus zu verlassen. Alle Änderungen haben eine direkte Auswirkung auf den gesamten Prozess.
- Wählen Sie **Änderungen verwerfen** aus, um die alten Ausgangswerte wiederherzustellen und den Bearbeitungsmodus zu verlassen. Dieser Vorgang hat keine Auswirkungen auf den Prozess.

1.8.5. Erneutes Ausführen eines fehlgeschlagenen Tokens

Wenn Sie im Feature **Prozessüberwachung** einen Prozess im Status **Fehler** sehen, können Sie den fehlerhaften Token zur Fehlerbehebung erneut ausführen. Beachten Sie die Fehlerursache in der Detailansicht eines Tokens, da Sie für die Fehlerbehebung möglicherweise die Variablen der Prozessinstanz ändern müssen.

Zum erneuten Ausführen eines Tokens navigieren Sie in die Detailansicht des Tokens. Wählen Sie **Erneut versuchen** aus und bestätigen Sie den Hinweis.

1.8.6. Erneutes Ausführen eines asynchronen Services

Bei der Verwendung eines asynchronen Services kann es beim Ausführen der Serviceaktivität zu einem Fehler bei der Kommunikation mit dem externen Dienst kommen. Der Fehler kann auftreten, wenn der Service-Endpunkt die Serviceanfrage zwar entgegennimmt, aber aufgrund von technischen Problemen nicht darauf antwortet. Um diesen Fehler zu beheben, können Sie in der Prozessüberwachung einen erneuten Aufruf des Services ausführen.

So geht's

1. Öffnen Sie im Feature **Prozessüberwachung** die betroffene Prozessinstanz und das Service-Token, das nicht beantwortet wurde.
2. Wählen Sie **Token verschieben** aus.
3. Wählen Sie die Zielaktivität für das Token aus und bestätigen Sie den Dialog.

1.8.7. Wissenswertes zur Protokollierung von Prozessen

Um den Durchlauf einzelner Prozessinstanzen nachverfolgen zu können, können Sie für die zugehörige Prozessversion die Protokollierung aktivieren.

Sie haben zwei Möglichkeiten, um das Protokoll anzuzeigen. In der Prozessüberwachung können Sie das Protokoll mit der Kontextaktion **Prozessprotokoll** anzeigen. Für Aufgaben, die Teil eines protokollierten Prozesses sind, können Sie die Protokollierung in der Perspektive **Prozess** in der Aufgabenübersicht anzeigen.

Unterschiedliche Ansichten

Für die Prozessprotokolle gibt es zwei verschiedene Ansichten:

- In der **fachlichen Ansicht** wird der Fokus auf Benutzeraktivitäten gelegt. Das Protokoll enthält in der fachlichen Ansicht Informationen zum Prozessstart und Prozessende, zu Benutzeraktivitäten, zu beteiligten Personen sowie zu eingegebenen Daten.
- In der **technischen Ansicht** werden alle durchlaufenen Aktivitäten der Prozessinstanz aufbereitet. Dazu zählen alle Elemente, die in der BPMN-Definition vorliegen, Serviceaufrufe, eingetretene Fehlersituationen und viele weitere technische Details. Die technische Ansicht eines Protokolls ist nur in der Prozessüberwachung verfügbar.

Aufbewahrung von Protokollen

Protokolle sind während der gesamten Laufzeit einer Prozessinstanz verfügbar. Nach Abschluss der Prozessinstanz können die Protokolle abhängig von der Konfiguration bis zu einem Jahr aufbewahrt werden. Nach Ablauf der Aufbewahrungsfrist werden die Protokolldaten gelöscht.

Export von Protokollen in ein d.3-Repository

Wenn Sie Protokolldaten über die Aufbewahrungsfrist hinaus aufbewahren möchten, können Sie die Protokolle in Ihrem d.3-Repository speichern. Zum Speichern der Protokolle im Repository müssen Sie zunächst den Protokollexport konfigurieren. Nach Abschluss einer Prozessinstanz wird das Protokoll anschließend automatisch in Ihrem d.3-Repository gespeichert.

Sie können die Protokolle nach dem Export weiterhin in der Prozessüberwachung und in der Aufgabenansicht anzeigen.

Besondere Prozessvariablen

Sie können dem Protokoll bei Benutzeraufgaben zusätzliche Informationen in Form einer Entscheidung und eines Kommentars hinzufügen. Dafür können Sie die Prozessvariablen **dv_decision** und **dv_comment** durch einen Bearbeitungsdialog übergeben.

Die Inhalte werden im Protokoll besonders hervorgehoben.

1.8.8. Aktivieren der Protokollierung von Prozessen

Beim Bereitstellen einer Prozessdefinition haben Sie die Möglichkeit, die Protokollierung zu aktivieren. Standardmäßig werden Protokolle für 30 Tage ab dem Prozessende aufbewahrt.

Sie müssen die Protokollierung für jeden Prozess einzeln einstellen.

So geht's

1. Öffnen Sie das Feature **Prozessadministration**.
2. Wähle Sie die Aktion **Neuen Prozess bereitstellen** aus.
3. Wählen Sie die BPMN-Datei aus, die Sie bereitstellen möchten.
4. Klicken Sie auf **Hochladen**.
5. Klicken Sie auf **Prozessprotokollierung**, um die Einstellungen für die Protokollierung zu öffnen.
6. Aktivieren Sie die Protokollierung.
7. Aktivieren Sie optional den Export für Protokolle in Ihr d.3-Repository, um eine revisionssichere Archivierung zu verwenden.
8. Ändern Sie bei Bedarf die Aufbewahrungszeit der Protokolle.
9. Bestätigen Sie den Dialog mit **Bereitstellen**.

Wenn Sie eine neue Version einer Prozessdefinition bereitstellen, werden die zuletzt verwendeten Einstellungen für die Protokollierung der Prozessversion übernommen. Wenn Sie die Einstellungen nachträglich ändern möchten, müssen Sie die Prozessdefinition erneut bereitstellen.

1.8.9. Erstellen und konfigurieren eines Prozessereignisses

Mithilfe von Prozessereignissen können Sie Prozesse über einen JSON-Endpunkt (z.B. via Webhook) starten. Erstellen Sie dafür zunächst ein Prozessereignis.

So geht's

1. Öffnen Sie das Feature **Prozessadministration**.
2. Wähle Sie **Prozessereignisse** aus.
3. Klicken Sie auf das Plussymbol, um ein neues Prozessereignis zu erstellen.
4. Geben Sie einen Namen für das zu erstellende Prozessereignis ein und klicken Sie auf **Hinzufügen**. Das Prozessereignis ist wird erstellt. Sie müssen dieses nun konfigurieren.
5. Wählen Sie im Auswahlfeld **Prozess** den Prozess aus, der gestartet werden soll.
6. Sie können optional **Business Key** oder **Correlation Key** für den Prozessstart definieren. Geben Sie hierzu im entsprechenden Feld einen Ausdruck an.
7. Im Bereich **Zuordnungen definieren** können Sie Daten aus dem JSON-Endpunkt zu Prozessvariablen zuordnen.
8. Klicken Sie auf das Plussymbol, um eine neue Zuordnung zu definieren.
9. Geben Sie einen Namen für die Zuordnung ein.
10. Geben Sie im Feld **Endpunktpfad** einen Ausdruck an, dessen Ergebnis beim Start in die Prozessvariable geschrieben wird.
11. Wählen Sie die **Prozessvariable** aus.
12. Klicken Sie auf **Hinzufügen**, um die Zuordnung durchzuführen.
13. Wiederholen Sie die Punkte 8-12 für alle Zuordnungen, die Sie definieren möchten. Beachten Sie, dass alle Pflichtvariablen des Prozesses zugeordnet werden müssen.
14. Speichern Sie das Prozessereignis.

In der Übersicht der Prozessereignisse können Sie die URL zum Auslösen eines Prozessereignisses jeweils im Drei-Punkte-Menü durch Klick auf **Endpunkt-URL in Zwischenablage kopieren** ermitteln.

Weitere Informationen zum Aufruf des Endpunktes finden Sie in der API-Dokumentation.

1.8.10. Wissenswertes zu Ausdrücken in Prozessereignissen

Sie können beim Erstellen von Prozessereignissen variable Ausdrücke für die Zuordnung von Werten aus dem JSON des Endpunktaufrufes verwenden. Die Ausdrücke verwenden die Syntax der Java Expression Language.

Ausdrücke dürfen lediglich folgende Elemente verwenden:

Element	Restriktion
Lesender Zugriff auf die Eigenschaften aus dem beim Aufruf übermittelten JSON**: input.getValue**	Der an die Methode übergebene String muss einem gültigen JSON-Path entsprechen (siehe Beispiele).
Erzeugen von Mehrfachwerten: **collections.from**	
Operatoren und Zeichen	Erlaubte Operatoren und Zeichen: +, -, *, /, %, =, !, <, >, &, |, ?, (,), [,].
Keyword	Erlaubte Keywords: div, mod, eq, ne, lt, gt, le, ge, and, or, not, empty.
Numerische Werte	
Zeichenketten	Zeichenketten müssen in " " oder ' ' eingeschlossen sein.

Beispiele

Beim Endpunktaufruf übermitteltes JSON

```
{
   "doc":{
      "caption":"Process description - 2021-11-17",
      "categoryId":"dv.fol.basis.Correspondence",
```

```
      "dateCreated":"2021-11-17T07:39:36.502+00:00",
      "dateLastAccess":"2021-11-17T07:39:36.502+00:00",
      "dateUpdated":"2021-11-17T07:39:36.502+00:00",
      "dateUpdatedFile":"2021-11-17T07:39:36.502+00:00",
      "fileExtension":"PDF",
      "fileName":"correspondence",
      "id":"KR00000227",
      "number":"KR00000227",
      "size":51500,
      "status":"F",
      "versions":[
         {
            "id":0,
            "status":"F",
            "dateCreated":"2021-11-17T07:39:37.000+00:00",
            "fileName":"KR00000227.1",
            "fileExtension":"PDF",
            "size":51500
         }
      ],
      "properties":[
         {
            "id":"dv.fol.basis.Subject",
            "name":"Subject",
            "index":2,
            "dataType":"STRING",
            "isMultiValue":false,
            "value":"Process description"
         },
         {
            "id":"dv.fol.basis.Date",
            "name":"Date",
            "index":50,
            "dataType":"DATE",
            "isMultiValue":false,
            "value":"2021-11-17"
         }
      ]
   },
   "fileDestination":"somwhere",
   "importOk":"1",
   "user":{
      "id":"A64C2BD6-8EF5-4C6D-B1BA-7F632A2AC3ED",
      "name":"Jane Doe",
      "d3Id":"jdoe"
   },
   "docType":{
      "id":"dv.fol.basis.Correspondence",
      "name":"Correspondence",
      "type":"DOCUMENT_TYPE"
   }
}
```


Beispiele

```
// Hello World
${"Hello World"}

// 3
${3}

// false
${3 > 4}

// Value of doc.status in JSON
${input.getValue("$.doc.status")}

// Value of a defined element in the doc properties in JSON.
// If an element is found, an array with one element is returned. Can NOT
be used in combination with string concatenation.
${input.getValue("$.doc.properties[?(@.id ==
\'dv.fol.basis.Subject\')].value")}

// Value of a defined element in the doc properties in JSON.
// If an element is found, its value is returned and may be used within
string concatenation. With this expression you have to ensure, that an
element exists.
${input.getValue("$.doc.properties[?(@.id ==
\'dv.fol.basis.Subject\')].value")[0]}

// creates a collection with "a", "b" and "c"
${collections.from("a", "b", "c")}
```

Es sind auch Mischformen möglich, sodass Sie Texte und variable Inhalte direkt miteinander kombinieren können.

```
dmsObject:///dms/r/yourRepositoryId/o2/${input.getValue("$.doc.id")}
```

1.9. Tipps und Tricks

Erfahren Sie mehr rund um Arbeitserleichterungen und nützliche Hinweise zu Funktionen.

1.9.1. Sichern des Schlüssels für verschlüsselte Variablen in Skripten

Sie ver- und entschlüsseln alle Variablen in Skripten mit einem einzigen Schlüssel. Der Schlüssel erscheint nach dem ersten Start von d.velop scripting im Arbeitsverzeichnis in der Datei **top.secret**. Ohne die Datei **top.secret** können Sie verschlüsselte Variablen in vorhandenen Skripten nicht mehr entschlüsseln. Beachten Sie folgende Empfehlungen:

- Sichern Sie die Datei **top.secret** abseits der vorhandenen Backup-Funktionen.
- Geben Sie nur dem ausführenden Benutzer (Benutzer, der d.velop process manager startet) Leserechtigungen für die Datei **top.secret**.

1.9.2. Aufrufen einer Schnittstelle mit eigener Certification Authority in einem Skript

Auf vielen On-Premises-Umgebungen existiert eine eigene Certification Authority (CA) für die Signierung von Serverzertifikaten. Oftmals sind diese CAs nicht von einer vertrauenswürdigen Stelle unterzeichnet, sodass die NodeJS-Ausführungsumgebung diesen Zertifikaten nicht vertraut. Dieser Umstand führt dazu, dass aus der Scripting-App heraus keine REST-Abfragen in Richtung dieser Systeme gesendet werden können.

Für die Ausführung von REST-Abfragen müssen Sie den öffentlichen Teil des CA-Zertifikates im PEM-Format importieren. Da die Zertifikate oftmals bereits im Zertifikatsspeicher von Windows gespeichert sind, können Sie den öffentlichen Teil des Zertifikats aus dem Zertifikatsspeicher exportieren.

> **Anmerkung**
> Das Zertifikat muss im PEM-Format sein. Exportieren Sie das Zertifikat mit dem Format "Base-64-codiert X509" oder konvertieren Sie das Zertifikat per openSSL. Anschließend können Sie die Datei bzw. den Inhalt der Datei als String im Skript verwenden.

Sie haben zwei Optionen, um selbstsignierte Zertifikate einzubinden:

Option 1

In der NodeJS-Ausführungsumgebung können Sie global eigene Zertifikate einbinden. Verwenden Sie die Systemumgebungsvariable **NODE_EXTRA_CA_CERTS=<Pfad zur PEM-Datei>** und starten Sie d.velop process manager neu.

Option 2

Fügen Sie Ihrem Skript die Abhängigkeit **undici** hinzu. Anschließend können Sie das Zertifikat von der Festplatte laden. Alternativ können Sie den Zertifikatsstring im Skript eintragen und dem **Agent**-Objekt als CA hinzufügen:

```
//Load undici agent and get dispatch function setter
const { Agent, setGlobalDispatcher } = require('undici')
module.exports = async (req, res) => {
        //Read cert file from filesystem
    const fs = require('fs');
    const certString = fs.readFileSync("C:\\certs\\test.pem", 'utf8')
        //Create a new agent using the loaded cert string as ca
    const agent = new Agent({
    connect: {
        ca:  certString
     }
    })
        //Set global dispatcher function
    setGlobalDispatcher(agent)
        //All regular fetch calls will now use the new agent
    const response = await fetch("https://my.secured.host/scripting", {
        // Adding method type
        method: "GET",
        // Adding headers to the request
        headers: {
            "Accept": "application/json"
        },
    })
    if(!response.ok) {
        console.log("Response was not ok")
    }
    const result = await response.json();
    console.log("Response from fetch", result)
    res.status(200).set("Content-Type", "text/plain").send(result);
}
```

1.9.3. Wissenswertes zum Erstellen von Skripten

Beim Entwickeln von JavaScript-Skripten ist es wichtig, bewährte Methoden zu verwenden, um den Code effizient und fehlerfrei zu gestalten.

In diesem Artikel finden Sie Informationen zur Verwendung der Methode **async/await** für bestimmte Abläufe. Wir erklären außerdem, warum Sie für Anfragen die Methode **fetch** anstelle von **axios** verwenden sollten.

Vereinfachte Handhabung von Abläufen mit der "async/await"-Methode

Beim Programmieren mit JavaScript können viele Aufgaben gleichzeitig erledigt werden. Obwohl es parallele Abläufe gibt, wird der ganzheitliche Ablauf nicht beeinträchtigt. Um parallele Abläufe zu steuern, ist die Verwendung der Methode **async/await** sehr hilfreich.

Tipps für die Verwendung der Methode "async/await":

- **Verwenden von "async"-Funktionen:** Verwenden Sie das **async**-Schlüsselwort, um Funktionen zu kennzeichnen, die asynchrone Aktionen ausführen. Der Code wird leserlicher, wenn Sie die **await**-Methode in diesen Funktionen verwenden.
- **Fehler behandeln mit "try/catch"-Funktionen**: Für eine ideale Fehlerbehandlung sollten Sie **await**-Anweisungen in **try**- und **catch**-Funktionen einfügen. So können Sie Fehler identifizieren und entsprechend darauf reagieren.

Beispiel

```
module.exports = async (req, res) => {

    try {
        let resp = await fetch("/identityprovider/scim/user");
        if (resp.ok) {
            console.log(await resp.json());
        }
    } catch (e) {
        console.error(e)
    }

}
```

Verwenden der "fetch"-Methode für Anfragen anstelle von "axios"

Wenn Sie Daten aus dem Internet abrufen möchten, benötigen Sie eine Möglichkeit zum Senden von Anfragen. Die Methode **axios** ist weit verbreitet, jedoch reduziert die Methode **fetch** die Anzahl der benötigten Abhängigkeiten. Die Methode **fetch** ist bereits in der Anwendung enthalten.

Tipps für die Verwendung der "fetch"-Methode:

- **Ähnliche Verwendung wie im Browser:** Die Verwendung der **fetch**-Methode ist ähnlich wie im Browser. Sie können die Methode verwenden, um Daten aus dem Internet abzurufen.
- **Reagieren auf Antworten**: Wenn Sie die **fetch**-Methode verwenden, bekommen Sie immer eine Antwort. Sie können die Methoden **await** und **fetch** kombinieren, um auf die Antwort zuzugreifen.

1.10. Häufig gestellte Fragen

Sie finden in diesem Thema Antworten zu häufig gestellten Fragen.

1.10.1. Warum tritt beim Einfügen einer Variablen ein unbekannter Serverfehler auf?

Wenn Sie eine Prozessvariable einfügen wollen, die bereits in einer anderen Groß-/Kleinschreibung existiert, kann es zu einem Fehler kommen. Der Fehler tritt auf, wenn die Sortierung der Datenbank

die Groß-/Kleinschreibung nicht beachtet und somit nicht zwischen den beiden Variablen unterscheiden kann.

Zur Vermeidung des Fehlers verwenden Sie beim Einfügen einer Variablen die bereits verwendete Groß-/Kleinschreibung.

1.10.2. Warum wird der Ladeindikator beim Speichern der Datenbankkonfiguration so lange angezeigt?

Beim Speichern der Datenbankkonfiguration kann es vorkommen, dass der Ladeindikator mehrere Minuten lang angezeigt wird. In diesem Fall haben Sie bei der Konfiguration die Daten möglicherweise nicht korrekt eingegeben. Sie müssen die Konfigurationsdaten für die Datenbankverbindung manuell löschen und erneut eingeben.

So geht's:

1. Beenden Sie die Anwendung über den Windows-Dienstemanager.
2. Löschen Sie die Datei **process-app-db.properties** im Verzeichnis **conf** unterhalb des Standardinstallationsverzeichnisses.
3. Starten Sie die Anwendung über den Windows-Dienstemanager.
4. Konfigurieren Sie die Datenbankverbindung im Feature **Konfiguration**.
5. Starten Sie die Anwendung neu.

1.10.3. Wie aktualisiere ich den JDBC-Treiber der Datenbank?

Wenn es ein Update zur verwendeten Datenbank gab, müssen Sie möglicherweise den JDBC-Treiber aktualisieren.

So geht's

1. Beenden Sie den Dienst der Anwendung.
2. Öffnen Sie den Ordner **lib** im Standardinstallationsverzeichnis der Anwendung.
3. Löschen Sie die Datei **pa-jdbc-***.jar** (die Sterne stehen für die Datenbankmanagementsystem-spezifische Endung)
4. Starten Sie den Dienst der Anwendung erneut.
5. Öffnen Sie im Feature **Konfiguration** den Eintrag **Datenbank** unter **Prozesseinstellungen**.
6. Klicken Sie auf **Treiber hochladen** und wählen Sie die JAR-Datei aus.
7. Starten Sie die Anwendung neu.

1.10.4. Wo finde ich eine Übersicht über alle administrativen Aktionen?

Wenn Sie im Feature **Prozessadministration** in die Perspektive **Aktionsübersicht** wechseln, erhalten Sie einen Überblick über alle laufenden, gestarteten, abgeschlossenen und fehlgeschlagenen Aktionen. Mit dem Papierkorb-Symbol können Sie abgeschlossene und fehlerhafte Aktionen manuell aus dem Protokoll entfernen.

1.10.5. Wo konfiguriere ich die Größe des Maximalspeichers für eine Prozessinstanz?

Sie können die Datei **process-custom.vmoptions** verwenden, um die Größe des Maximalspeichers zu konfigurieren. Die Datei **process-custom.vmoptions** wird aus der Datei **process.vmoptions** referenziert. Im Unterschied zur Datei **process.vmoptions** bleibt die neue Datei **process-custom.vmoptions** bei Updates unverändert und Ihre Anpassungen bleiben bestehen.

Wenn Sie einen identischen Parameter in beiden Dateien angeben, wird der Inhalt der Datei **process-custom.vmoptions** bevorzugt, da diese Datei später geladen wird.

1.11. Weitere Informationsquellen und Impressum

Wenn Sie Ihre Kenntnisse rund um die d.velop-Software vertiefen möchten, besuchen Sie die digitale Lernplattform der d.velop academy unter https://dvelopacademy.keelearning.de/.

Mithilfe der E-Learning-Module können Sie sich in Ihrem eigenen Tempo weiterführende Kenntnisse und Fachkompetenz aneignen. Zahlreiche E-Learning-Module stehen Ihnen ohne vorherige Anmeldung frei zugänglich zur Verfügung.

Besuchen Sie unsere Knowledge Base im d.velop service portal. In der Knowledge Base finden Sie die neusten Lösungen, Antworten auf häufig gestellte Fragen und How To-Themen für spezielle Aufgaben. Sie finden die Knowledge Base unter folgender Adresse: https://kb.d-velop.de/

Das zentrale Impressum finden Sie unter https://www.d-velop.de/impressum.